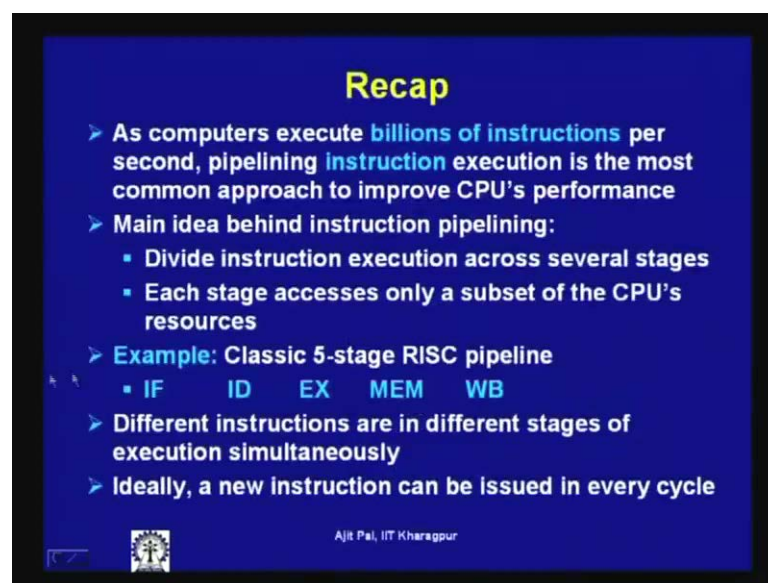


**High Performance Computer Architecture**  
**Prof. Ajit Pal**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 8**  
**Pipeline Hazards**

Hello and welcome to today's lecture on instruction pipeline and particularly, we shall focus on Pipeline Hazards in this lecture.

(Refer Slide Time: 01:04)



**Recap**

- As computers execute **billions of instructions per second**, **pipelining instruction execution** is the most common approach to improve CPU's performance
- Main idea behind instruction pipelining:
  - Divide instruction execution across several stages
  - Each stage accesses only a subset of the CPU's resources
- **Example: Classic 5-stage RISC pipeline**
  - IF    ID    EX    MEM    WB
- Different instructions are in different stages of execution simultaneously
- Ideally, a new instruction can be issued in every cycle

Ajit Pal, IIT Kharagpur

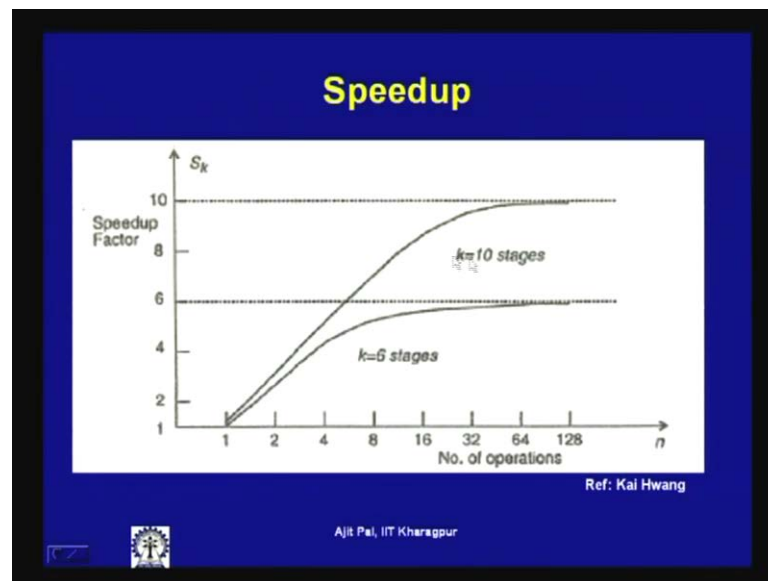
And briefly we can recapitulate what we discussed in the last lecture, you may recall that I mentioned that as computer execute billions of instructions per second, pipelining instruction execution is the most common approach to improve CPU's performance. So, since that is the primary job of a processor, each and execute instructions, instruction pipelining is profitable and widely used, since 1980's. And I mentioned that main idea behind pipelining, instruction pipelining is to divide instruction execution across several stages that means, we divided into subtasks. And then, across different stages is divided, an stage accesses is only a subset of the CPU's resources, that I have elaborated in details.

And particularly, we have considered a classic 5-stage RISC pipeline as an example, where the pipeline is divided into 5-stages instruction fetch, instruction decode, execute, memory read or write and then, write back. And different instructions are in different

stages of execution simultaneously, and how it happens and how in the help of pipeline in register this is maintained, I have discussed in detail. And we have also seen that if the ideal conditions are satisfied, then it is possible to input one instruction at a time or produced result for one instruction at a time.

So, it gives you a CPI is equal to 1, cycles for instruction that is equal to 1 in the ideal situations.

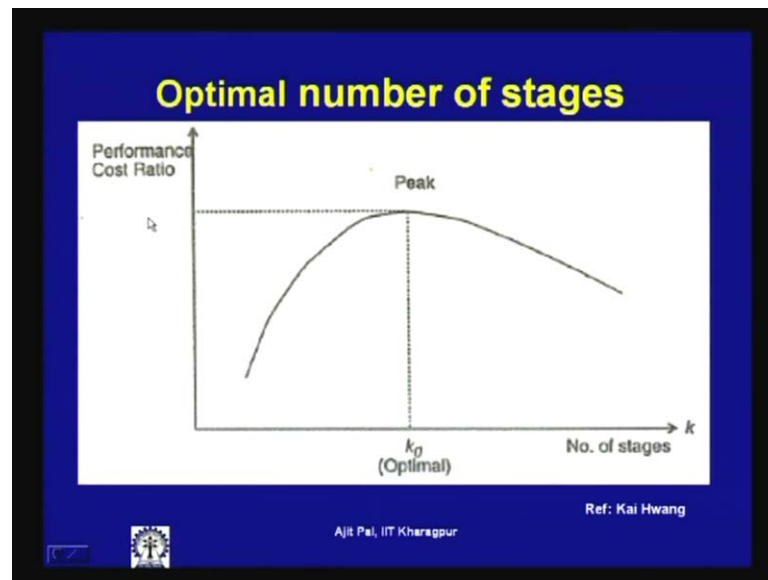
(Refer Slide Time: 03:07)



Question naturally arises, whether this is true for possible for situations and I mentioned about a parameters speedup, and we have seen only for executing large number of operations, then only you reach the ideal speedup. And as you can see that, if you got great stages, then if the number of operations is very few, then you can see the speedup factor is initially 1, if the number of operation is only one, if the number of operation increases.

In our case number of instructions is increases to executed, then you can see the speed of increases gradually and in attains a value of  $k$ , that is the number of stages only if any how got large number of instructions getting executed. So, is true for any value of  $k$ , for  $k$  is equal to 10, this is same is the situation and we have also discussed about the limits on a pipeline stages.

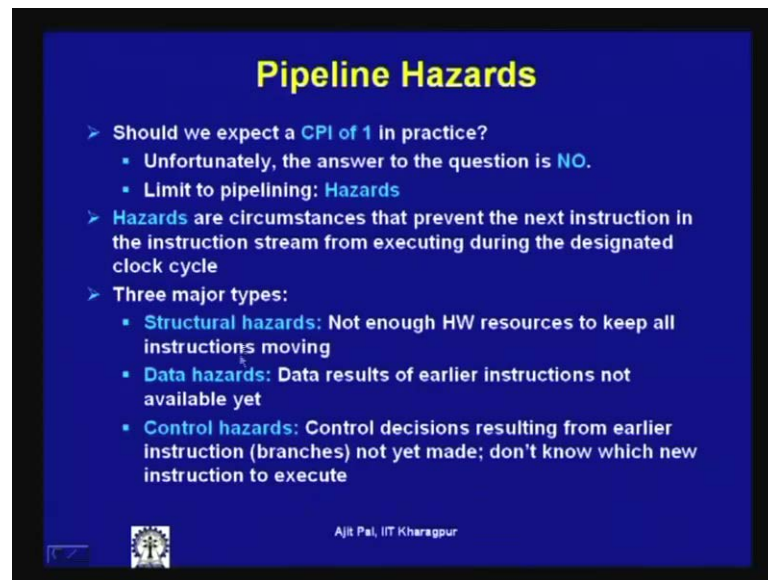
(Refer Slide Time: 04:14)



The number of stages if you increase it helps to you in improving the performance, because speedup increases, but as a increase the number of stages we have seen, we have two had pipeline resister as you add a new stage. So, that is a overhead as you keep on adding number of stages, you have to keep on adding the pipeline registers as well and as you keep on adding the pipeline resisters with the number of stages, the overheads keeps on increasing. And as a consequence we can see the cost performance, the performance cost ratio does not really increased linearly as the value of  $k$  increases.

So, it increases and then, attends the peak value and if you further increase the number of stages, then the performance is degrades and thus actually puts a limit on the number of stages that a pipeline processes should have. And this particular optimum value are being tried in modern assessors, and the number of stages which is increase to as many as possible at obviously, there is a upper limit beyond which does not give you good results. So, the resources keeps on increasing, as the number of  $k$  increases resource requirement keeps on increasing.

(Refer Slide Time: 05:56)



### Pipeline Hazards

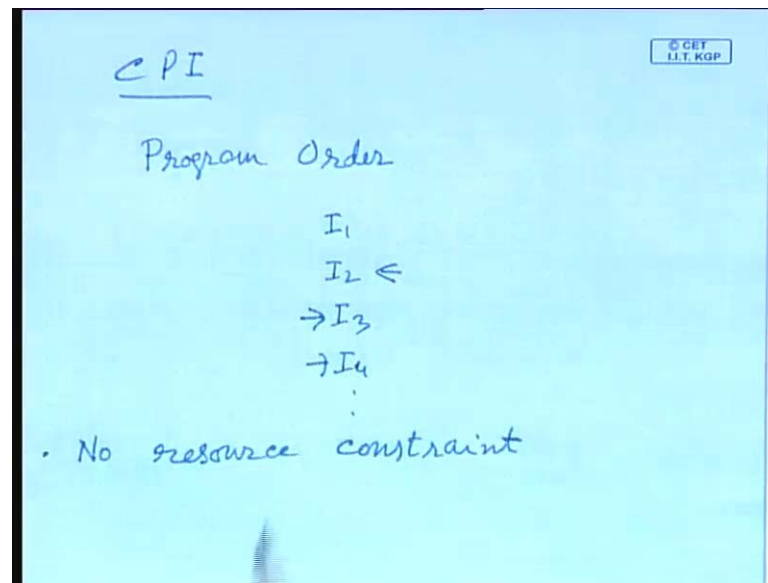
- Should we expect a **CPI of 1** in practice?
  - Unfortunately, the answer to the question is **NO**.
  - Limit to pipelining: **Hazards**
- **Hazards** are circumstances that prevent the next instruction in the instruction stream from executing during the designated clock cycle
- Three major types:
  - **Structural hazards**: Not enough HW resources to keep all instructions moving
  - **Data hazards**: Data results of earlier instructions not available yet
  - **Control hazards**: Control decisions resulting from earlier instruction (branches) not yet made; don't know which new instruction to execute

Ajit Pal, IIT Kharagpur

Now, today we shall focus on pipeline hazards, question arises should you expect a CPI of 1 in practice, in CPI is 1 then it is the ideal situation, ideal conditions are made. What were the ideal conditions, number 1 ideal condition was that the instructions are independent, they can be executed independent of each other. And only then, we can we can achieve a CPI is equal to 1, but unfortunately we cannot have complete independent instruction cannot be fully independent.

And so we cannot really get a CPI of 1 and there is some limit and that limit is imposed by hazards, so we shall discussed about hazards. What is a hazards, hazards are circumstances that prevent the next instruction in the instruction stream from executing during the designated clock cycle, we have fetching in our program order.

(Refer Slide Time: 07:12)



We have a sequence of instruction, instruction 1 followed by instruction 2 followed by 3 followed by instruction 4 and so on, so now we would like to execute this instruction one of the other. And whenever you do pipelining we are obviously, some kind of overlap execution, now whenever you were try to do it in this way, you will find that where  $I_3$  should have been executed. We are not able to do because of hazard or were  $I_4$  is supposed to be executed, cannot be executed, because of dependence among the instructions, and that is known as hazards.

And the hazards can be broadly divided into three major types, what are the three major types, number 1 is structural hazards. And structural hazards arises when you do not to have enough resources to keep all instructions moving, initially never ideal situation we will assume that there is no resource constraint. But, this cannot be nearly true in practice, because always there will be some resource constraint, because you would like to optimize the performance or a given cost.

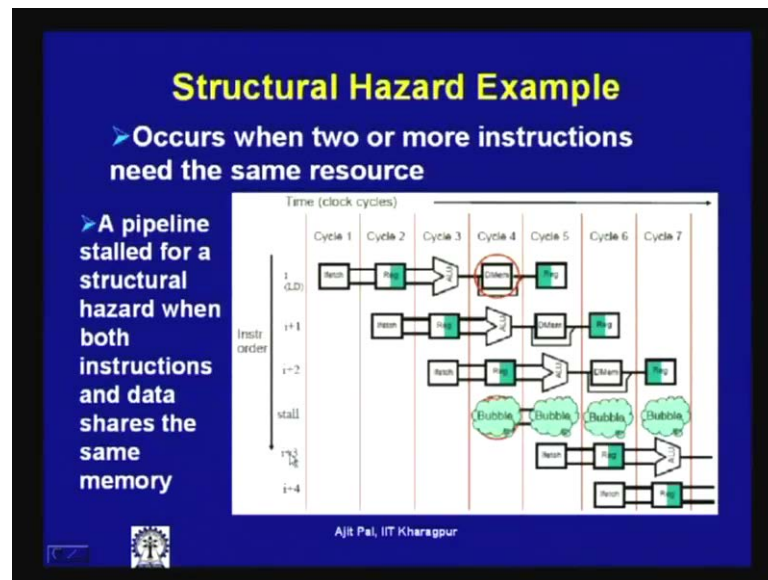
So, cost we will keep on increasing as you proved a additional resources in the system, so there will be a structural hazard will arise, will not arise if you have unlimited resources. But, whenever you impose on restriction on the resources, then structural hazard arises how, let me I shall explain with the help of example, but before that let me explain the other two types of hazards. Second type of hazard is known as data hazards, so data hazards arises, because data results of earlier instructions not available yet.

That means, what can happen suppose these are the instructions situation which you are executing one after the other, that is instruction 2 may require some data which is produced by instruction I 1. And whenever I 2 is trying to read that data, then I 1 is not yet written into its register. So, we have seen that primary means by which the data flows taking place is two registers, because all the ALU operations are involving registers, so in case of a processor. So, if the result is not yet written into the registers, then if the same data is needed by a sub-sequentially instructions, then this data hazard will occur.

((Refer Time: 10:25)) Third type of hazard is known as control hazard, control decisions with a result from earlier instruction not yet made, so do not know which new instruction to execute. So, whenever you will find that, there is an if-then type of statements, present in your program, now that condition for fetch I mean condition on a fetch that branching decision, is something is true then it will execute a consequence. If that condition is not true, then it will execute another consequence, but if the condition is not yet known, then fetch consequence and executed.

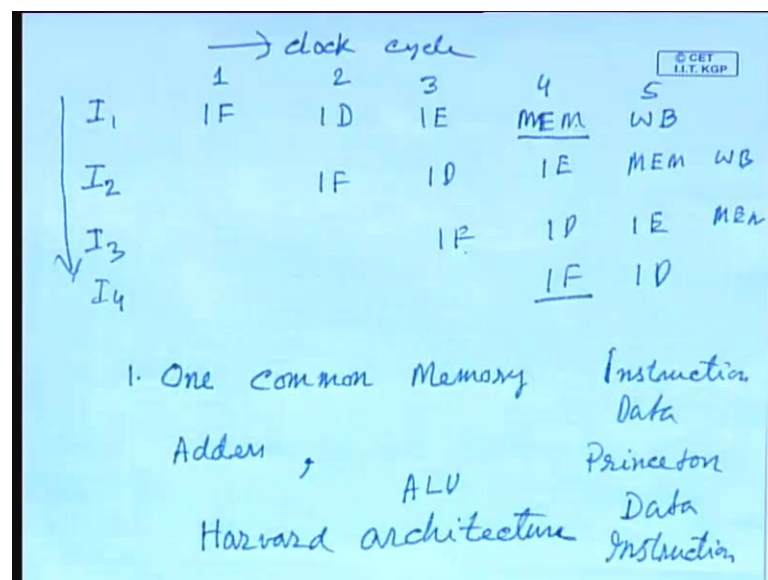
Because, if the consequences to be executed is known in the processor can fetch from the memory, that particular consequence, but if that is known then processor needs to wait till the decision is known; and that leads to what is known as control hazards. So, I shall elaborate these hazards one of them be other, first let's focus on the structural hazard.

(Refer Slide Time: 11:39)



The structural hazards occurs when two or more instructions need this same resource, we have seen that an instruction is divided into say classically high stages.

(Refer Slide Time: 12:00)



Say I 1 you have brought instruction fetch, instruction decode, then instruction execution, then memory operation, then wrote back, so this is for instruction 2 we are executing in a overlap manner. So, this cycle 1, this is cycle 2, this is cycle 3, 4, 5, then instruction fetch, instruction decode, instruction execution, memory operation and write back. So, this is how it happens, instruction fetch, instruction decode and instruction

execution, the memory operation and so on; so this is the how the instruction are executed in this way and in this relation we have bought the clock cycle or time.

Now, we find that whenever we go to this particular point, see you abort instruction 4, instruction fetch and instruction decode is going on, so if you look at this point, here we are performing from memory operation, and also you are performing instruction fetch. Now, if you have got one common memory, then what will happen the these two operations and cannot be performs simultaneously, this particular I mean instruction 1 will be writing some data into a, I mean it will involve memory that means it will access memory, maybe load or store whatever it may be.

And here ((Refer Time: 13:53)) the instruction 4 is doing fetched that is also stored in memory that means, for instruction and data we have common memory, then there will be a hazard here. Because, in may recall that, we earlier processor like in prince and architecture proposed by ((Refer Time: 14:24)), there we had got only one memory, common memory. So, if you have got only one memory then, there will be a structural and hazard at this point, similarly the example here we were doing instruction execution and instruction decode.

And instruction execution may involve some computation, it will always involve some kind of computation. Now, whenever you are doing the performance instruction fetch and instruction execution, you are doing instruction execution and instruction fetch and instruction decode, in such cases also you will see, he will require some address to generate the next address, whenever you are doing instruction fetch your calculating the next address. And whenever you performing instruction execution, you are using the adder, so in these two cases if you use the same ALU to perform both the thing.

That means, calculation of the next address as well as, I mean computation of the operation additions, subtraction whatever it may be, then again there will be a hazards and that is the reason why you may have noticed in our simple pipeline, we had got separate adder. And also in addition to that the arithmetical logic unit in, so it is not done by this same in, so what we are doing, we are adding additional resources for that this happens.

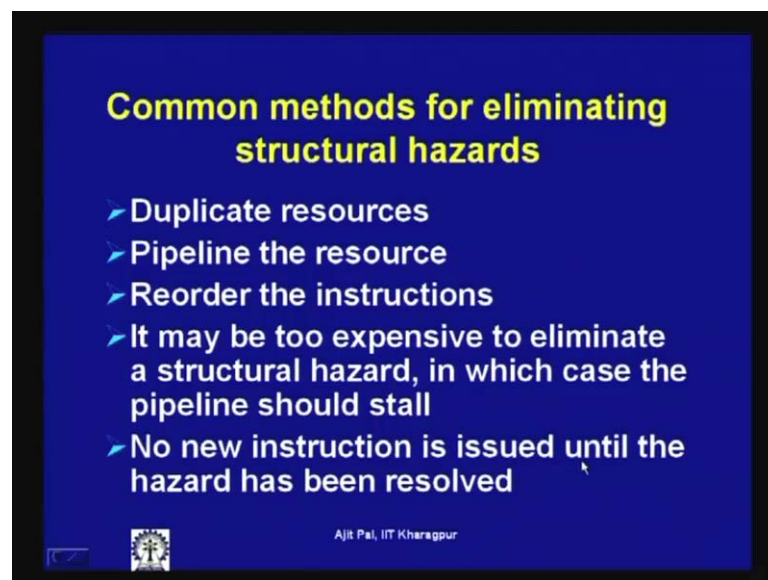
So, in this particular diagram, we have seen the that if you have a shared memory then you have to know their alternatives, but to stall the execution. In other words, what will



do that instruction fetch of the fourth instruction is delayed till the 5th cycle and you are losing one instruction side. So, one cycle is lost whenever you have got a single memory and that is the reason why that Harvard architecture is used, Harvard architecture where you have got two separate memories, one for data and another for instruction.

So, whenever we have got a two separate memories no problem, this particular operation will involve the data memory, accesses of data memory and this particular, here if you fetch the instruction, then it will involve program memory, so the hazard is overcoming. So, this is how with the help of additional resources you can overcome the structural hazards, so pipeline stalls for a structural hazards when both instructions and data are the same memory shown in this particular diagram.

(Refer Slide Time: 17:35)



Now question is, what are the different ways by structural hazards can be overcome, number 1 is duplicate resources as we have already seen, we are duplicating the resources adding both memories, adding a adders, such that the structural hazards that is overcoming. Another alternative way of overcoming structural hazard is to pipeline the resource, so a particular resource may be pipeline, so if you can pipeline a particular resource it can be an ALU, it can be some other resource memory.

If you can pipeline it is operation, then what you can do since in a pipeline system you have seen different operations can be performed for different tasks in a overlapped manner, so then also it is possible to overcome structural hazards. But, all resources

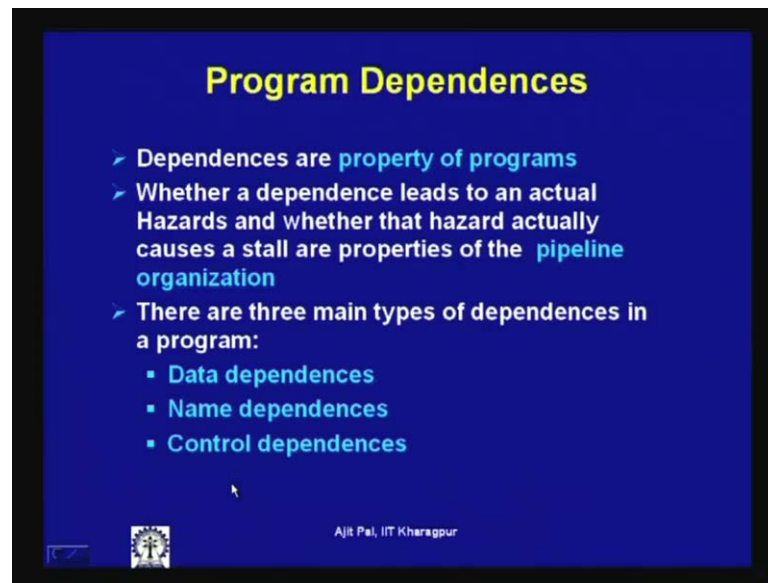
cannot be pipeline unfortunately, so if a particular resource can be pipelined, then instead of duplicating, you will go for pipelining the resource that will be more cost-effective in the sense pipelining involves much lesser cost, then duplicating a resource.

So, this is the better alternative, but that alternative may not be always possible and not, that is why duplicating resources is commonly used, because it is simple easy to implement and always possible. Then reorder the instructions, so this is the third alternative, where whenever you are facing some hazard for example, if there is any way by fetch two instructions or causing some kind of hazard. So, what you can do the instruction sequence can be changed that, you can reorder the sequence of instructions.

So, without affecting the result that means, the without affecting the program flow, program flow means that is intended that was intended in your source program, whatever outcome was expected, that output should be obtain, without violating that, if you can reorder the instructions and overcome the hazard that is also possible, so you can reorder the instructions. And it may be too expensive to eliminate a structural as hard in which case the pipeline should stall, so the last resort is to stall the pipeline.

So, whenever none of ((Refer Time: 20:31)) physical duplicating resource is very expensive, pipelining is not feasible, reordering of instructions is not feasible, in such case you have no the alternative, but to stall the instruction for a cycle and that will definitely overcome the structural hazard. So, whenever you are doing this, no new instruction is issued until the hazard has been resolved, so in the previous case we have seen, just by introducing one stall, we are able to overcome the structural hazard for our 5-stage pipeline. So, we have discussed about the structural hazard, now let us focus on data hazard.

(Refer Slide Time: 21:24)



So, what is the primary reason for data hazard that we shall explore our, as I mentioned our earlier assumption that instructions are independent is not true, there will be dependences. So, dependences are the hoop cause of data hazard, so we shall do the analysis of different types of dependences to understand, why data hazard occurs and how it occurs. So, first of all you should understand these two aspects, number 1 is dependences are property of programs that means, that property of program is independent of the machine implementation.

A program has been written, a source code is there and that source code as incorporated some dependences among different instructions, so this is independent of the processors. So, that part is the property of programs, that cannot be changed by whenever you go for implementation, secondly whether a dependence leads to an actual hazards and whether the hazard actually causes a stall or properties of the pipeline organization. That means, these two aspects are different, one is the property of the programs, another is the outcome of the pipeline organization.

That means, whether a particular dependence will lead to some hazard and stall, that will depend how the pipelining is done. And there are three main types of dependences in a program, we shall explore three main types of dependences, number 1 is known as data dependences, second is known as name dependences, third known as control

dependences. So, let us try to understand these few different types of dependences that is inherently present in a program.

(Refer Slide Time: 23:39)

### Data Dependences

➤ An instruction  $j$  is **data dependent** on instruction  $i$ , iff either of:

- **Direct:** Instruction  $i$  produces a result that is used by instruction  $j$
- **Transitive:**
  - Instruction  $j$  is data dependent on instruction  $k$  and
  - Instruction  $k$  is data dependent on instruction  $i$
- **Dependence within a single instruction (such as ADD R1, R1, R1) is not considered as a dependence**

Ajit Pal, IIT Kharagpur

So, first we shall focus on data dependences, so an instruction  $j$  is data dependent on instruction  $i$ , if either of the following conditions hold, number 1 is there is direct that a dependence. Instruction  $i$  produces a result that is used by instruction  $j$  as it is shown in this particular code  $r1$  of operation  $r2$ , you are performing some arithmetic operation, arithmetic logical operation on the content of  $r1$  with that of the content of  $r2$ ; and you are storing the result in resistor  $r3$ .

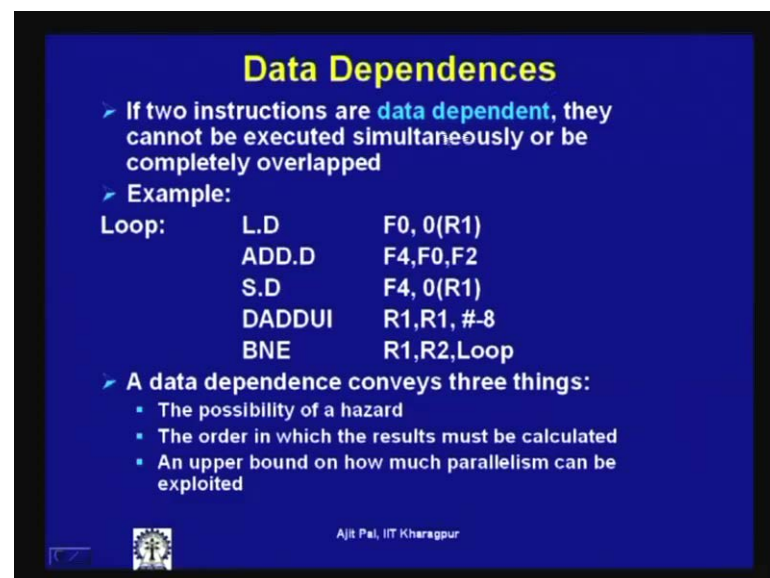
Now, that resistor  $r3$  is being used as the source of operation, in case in the second instruction where you are performing in the content of  $r3$  operation, content of  $r4$  and the result is being stored in  $r5$ . So, in this case you can see there is a direct data dependence, that instructions 2 this data dependent on the instruction 1, so or you can say if this is the  $j$  and this is the  $i$ , instruction  $j$  is data dependent on instruction  $i$ . So, this is the direct that the dependence and it may not be always direct data dependence, and may not be always present, there can be data dependence can be transitive is nature.

So, in such a case instruction  $j$  is data dependent on instruction  $k$  and instruction  $k$  is data dependent on instruction  $i$ , as it is illustrated with the help of this example, here as you can see. That instruction 2, second instruction is data dependent first instruction, but the third instruction is not data dependant on first instruction directly, but you can see the

second instruction is data dependent on the first instruction and third instruction is data dependent on the second instruction.

So, there is a kind of transitive relationship on data dependency and here, you can say that this r 5, that is third instruction is also data dependent on first instruction, may not be directly, but there is a transitive data dependence. However, dependence within a single instruction such as ADDDD, ADDD double R 1, R 1 is not considered as dependence, so this cannot be considered as dependence. Because, in this case all the three operations are involving same resistors, and this is not treated as a dependence.

(Refer Slide Time: 26:34)



**Data Dependences**

- If two instructions are **data dependent**, they cannot be executed simultaneously or be completely overlapped
- Example:  
Loop:    L.D        F0, 0(R1)  
          ADD.D    F4, F0, F2  
          S.D       F4, 0(R1)  
          DADDUI   R1, R1, #-8  
          BNE      R1, R2, Loop
- A data dependence conveys three things:
  - The possibility of a hazard
  - The order in which the results must be calculated
  - An upper bound on how much parallelism can be exploited

Ajit Pal, IIT Kharagpur

Now, let us come to another example, if two instructions are that a dependent, they cannot be executed simultaneously or be completely overlapped. Why it is necessary to analyze data dependence is very clear, because whenever we try to implement pipelining, then try to execute in a overlapped manner, that will lead to hazard and we have to stall. That is the reason why, if two instructions are data dependent they cannot be executed simultaneously or be completely overlapped, either they cannot be completely overlapped or simultaneously they cannot be executed.

So, let us look at this program loop L D, F 0, 0 (R 1), here as you can see the F 0 is the source of operand in the next instruction, so there is direct data dependence between these two, and also there is data dependence between first and third. So, floating point operations are being performed by this operation, and in the first instruction you are

loading, loading a particular operand a scalar value into a register F 0. Then you are performing addition if the content of a register F 2 and storing the result in F 4, and again here also you are storing the value, another data that is F 4 is being stored in the moment, that is the instruction that is being performing.

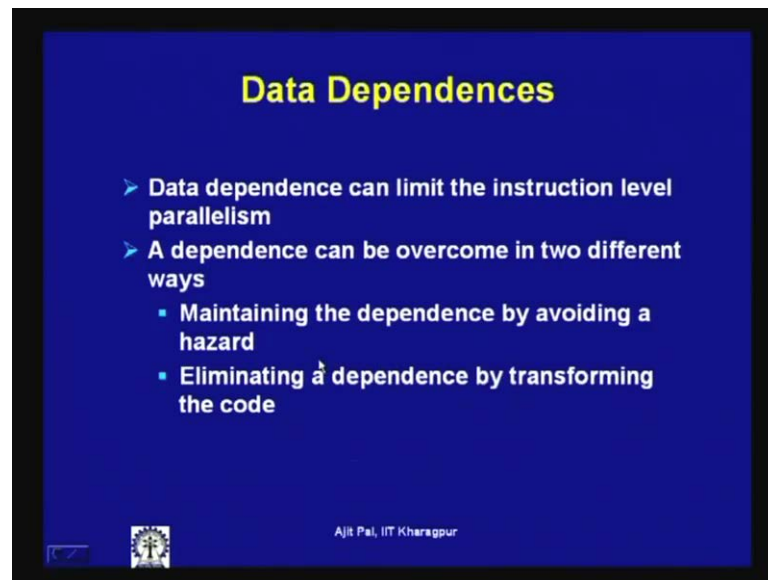
And here also you can see that R 1, R 1, R 2 R 1, R 8 and here a branch not equal here also there is data dependence, because you are performing some kind of subtraction, so that the it is actually as a pointer. So, it points to be next element and that is the reason why it is being done, and the next instruction bunch of operation is dependent on that, so there is a dependency here. So, in this particular loop you find that, there are several data dependence as present, even within this straight line code that is present and in this program.

So, a data dependence conveys three things, why it is necessary to analyze data dependence, because a data dependence will lead to three things or a conveys three things, that first of all there is a possibility of hazard. So, if there is data dependences there is a possibility of hazard, second is the order in which the results must be calculated that means, the order in which the instructions are appearing, which is known as program order, that program order needs to be maintained. And data dependence also ensures that the maintenance of this program order and this data dependence also gives an idea of the upper bound, on how much parallelism can be exploited.

So, later on we will see, we shall not only restrict to pipelining, we shall go for other type of processing like superscalar architecture, PLIW architecture where there will be multiple processing elements. So, where we shall be trying to exploit instruction level parallelism in little different way, so there also we will find that you will be executing some operations in parallel. Now, what kind of instruction level parallelism in addition to pipelining, whenever we shall go for other types of instruction level parallelism, what is the degree of parallelism that is present in a program, inherent in the program.

That will decide the order of this upper bound on how much parallelism that is present, so this data dependence analysis is very important, because of these three issues.

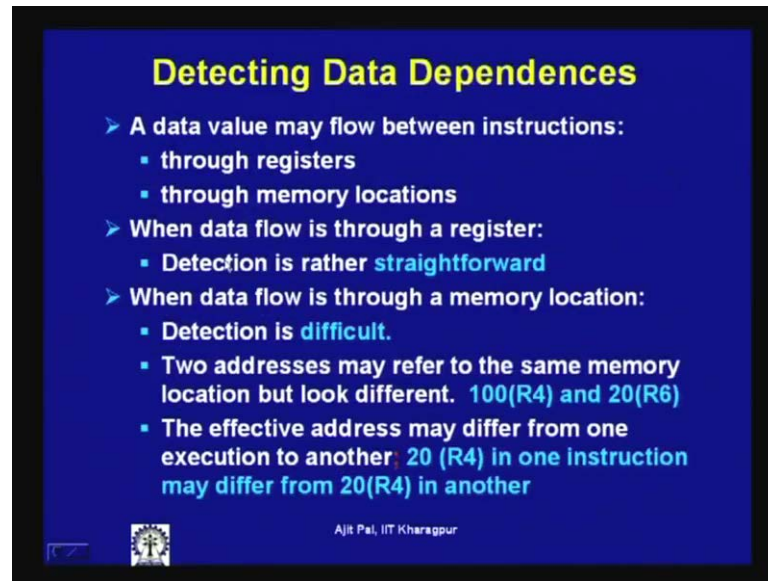
(Refer Slide Time: 31:16)



So, as I mentioned data dependence can limit the instruction level parallelism, a dependence can be overcome in two different ways. Now, suppose you have identified that there is data dependence, so in such a case what are you going to do, there are two possible alternatives, number 1 is you will maintain the dependence by avoiding a hazard. So, you will maintain the dependence and you will also try to avoid the hazard somehow, how that we shall discuss in detail a little later.

So, this is one possibility that without changing the data dependence, you will somehow avoid data hazards or avoid stalls that is one possibility. Second alternative is that he will modify the code itself that means, he will eliminate a dependence by transforming the code you will modify the program such that, the data dependence which was existing is eliminated. So, thereby avoiding hazards and stalls which is outcome of the data dependence.

(Refer Slide Time: 32:45)



**Detecting Data Dependences**

- A data value may flow between instructions:
  - through registers
  - through memory locations
- When data flow is through a register:
  - Detection is rather straightforward
- When data flow is through a memory location:
  - Detection is difficult.
  - Two addresses may refer to the same memory location but look different. 100(R4) and 20(R6)
  - The effective address may differ from one execution to another. 20 (R4) in one instruction may differ from 20(R4) in another

Ajit Pal, IIT Kharagpur

Now, question is how do you really detect data dependence, so data dependence may flow between instructions in two ways, number 1 is through registers, number 2 is through memory locations. So, by looking at the program you can by observing the program, you can detect data dependence, if same resistor is involved of if some memory address is involved. So, whenever same resistors involved through which data flow is taking place, there it is very easy to detect data dependence that means, when data flow is through a register, detection is rather straight forward.

On the other hand, when data flow is through a memory location detection is very difficult, detection cannot be done easily, whenever it is through registers, but let us considered two different examples why it is difficult. Two addresses may refer to the same memory location, but look different for example, we have used two into separate instructions.



(Refer Slide Time: 34:03)

Handwritten notes on a blue background showing effective address calculations. The notes include the following expressions and text:

- $100(R4)$
- $EA_1 = [R4 + 100]$
- $20(R6)$
- $EA_2 = [R6 + 20]$
- $EA_1 = EA_2$
- $20(R4)$
- $20(R4)$
- Data from Memory

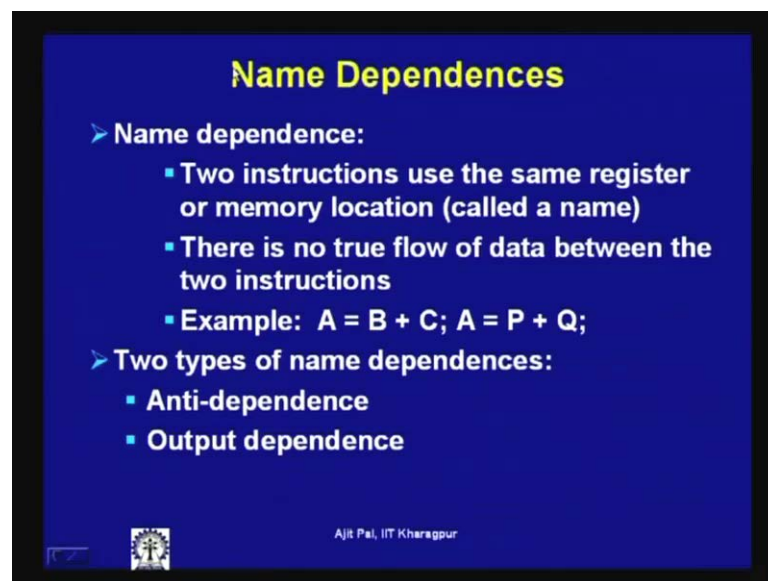
In one instructions we have used this R 4, this is the address that is being generated that means, in this case effective address is equal to content of R 4 plus 100, that is the effective address. So, this is how the effective address is calculated, and another instruction is that will where the effective address, is the instruction is this operant this is specified by this. So, in this case the effective address is equal to content of R 6 plus 20, now it may so happen that this effective address 1 and the effective address 2 maybe same.

So, if we analyze the program, then you can see although the addresses generated are same, by observing the program it is not possible to say that the effective address will be same. The reason for that is the value which will be present in R 4 and value that is present in R 6, can be dynamically modified, while executing a program, because memory locations the effective address that will be generated by calculating with the content of a resistor. And some displacement that is provided as part of the instruction, that cannot be observed, that cannot be detected by simply by observing the program.

Similarly, there is another example the effective address may differ from one instruction to the other, so in the second case for example, the in two difference instances in the instructions sequence, you have be same resistor in and same this place. Now, what can happened in between the resistor value may have been changed, so although these two it may appear to that these two will produce the same effective address.

But, in reality they may not generate the same effect at this, the reason for that is that the value of R4 may be dynamically changed while executing instructions. So, in this case we find that whenever the data dependencies are true, I mean the data flow takes place through memory, then it is difficult. So, later on we shall discuss about how the detection can be done, whenever it involves in false sharing, but whenever it is involving registers it is rather easy to detect.

(Refer Slide Time: 37:17)



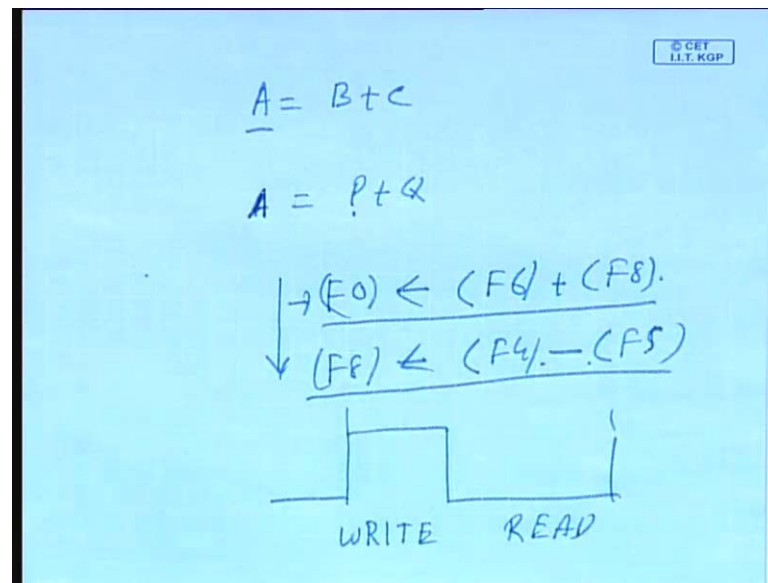
**Name Dependences**

- **Name dependence:**
  - Two instructions use the same register or memory location (called a name)
  - There is no true flow of data between the two instructions
  - Example:  $A = B + C$ ;  $A = P + Q$ ;
- **Two types of name dependences:**
  - Anti-dependence
  - Output dependence

Ajit Pal, IIT Kharagpur

Now, the second type of dependency is known as the name dependencies, so name dependencies are two instructions are same register are memory location called a name. So, there is no true flow of that data between the two instructions, we have taken of an example.

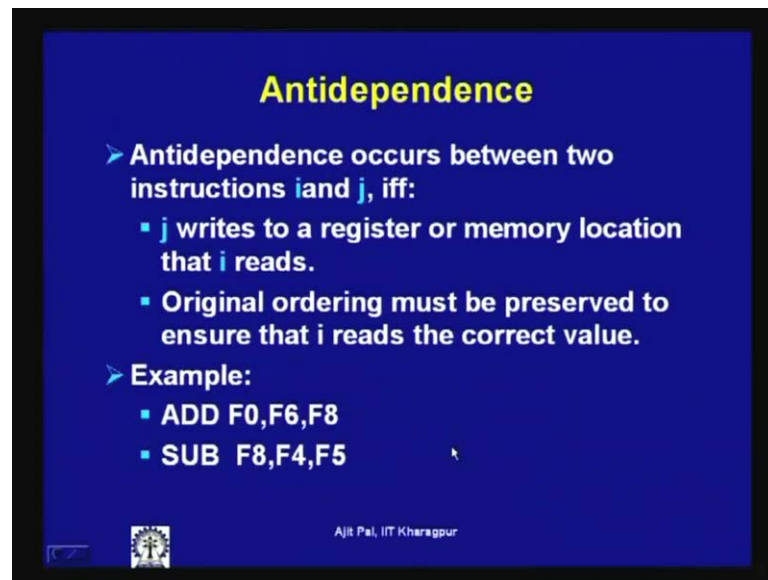
(Refer Slide Time: 37:46)



Let us consider one examples is say A is equal to B plus C and another is A is equal to P plus Q, now here what you are doing you are having the value taken from, maybe memory location were these store and another memory location where C store. And then, you are storing can be memory A, similarly here you have got another instructions, where you are storing in same memory, but the operands P and Q are different. Now, this particular case, there is no data dependency in the sense, there is no flow of data from one instruction to another.

But, this will also lead to a kind of dependence and there are two types of dependences, one is known as anti-dependence and other known as output dependence.

(Refer Slide Time: 38:47)



**Antidependence**

- Antidependence occurs between two instructions  $i$  and  $j$ , iff:
  - $j$  writes to a register or memory location that  $i$  reads.
  - Original ordering must be preserved to ensure that  $i$  reads the correct value.
- Example:
  - `ADD F0,F6,F8`
  - `SUB F8,F4,F5`

Ajit Pal, IIT Kharagpur

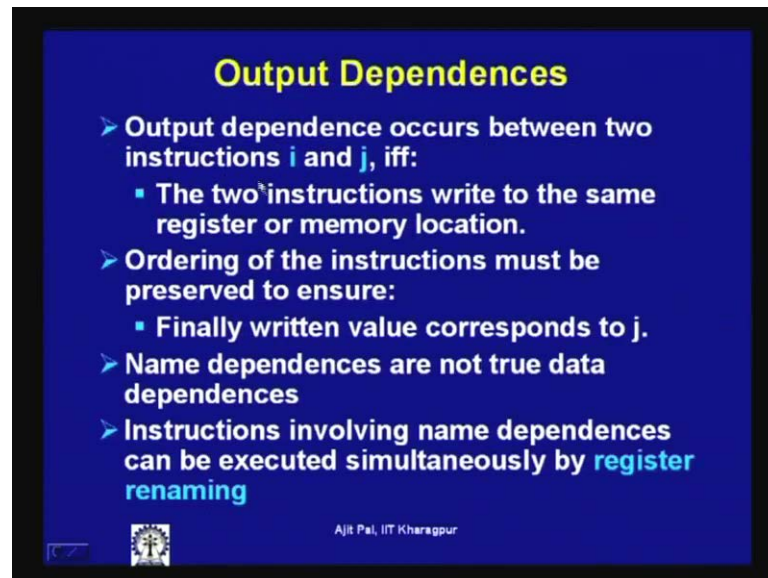
So, let us what are these two dependences, what is anti-dependence and what is output dependence. So, anti-dependence occurs between two instructions  $i$  and  $j$ , if  $j$  writes to the register or memory location that  $i$  reads, original ordering must be preserved to ensure that  $i$  reads the correct value. So, here as you can see in the first case, you are reading an operand from register 8 and then, performing this operation and the content of F 6 you are adding with content of F 8, then storing will result in F 0.

Now, you have got another instructions, which is following the first instructions, where you are storing the result in a F 8, I mean F 8 is equal to content of F 4 plus content of a F 5. So, in the first case what is happening F 0 is where you are storing F 6 plus F 8 and the second one, this is your addition and in the second case, you are performing F 8 is equal to F 4 minus F 5. So, this is what you are doing now, this is the program order now it may so happened that, this particular instructions who will executed before this instructions.

So, if this happens then what will happened then whatever value was intended to be read by the first instructions you will get a different value because it has been changed by subsequent instruction. So, subsequent instruction has modified it and then you are reading it, so will not get the correct result, so this is called the anti-dependence. So, this anti-dependence is arising whenever that program order is not being followed strictly and

that will happen particularly later on we shall see in superscalar architecture this can happen. So, this is known as anti-dependence

(Refer Slide Time: 41:15)



**Output Dependences**

- Output dependence occurs between two instructions  $i$  and  $j$ , iff:
  - The two instructions write to the same register or memory location.
- Ordering of the instructions must be preserved to ensure:
  - Finally written value corresponds to  $j$ .
- Name dependences are not true data dependences
- Instructions involving name dependences can be executed simultaneously by **register renaming**

Ajit Pal, IIT Kharagpur

Now, let us consider the output dependence, so output dependence occurs between two instructions  $i$  and  $j$  if and only if the two instructions write to the same register or memory location. Now, the ordering of the instructions must be preserved to ensure such that you know whenever in the same register or memory location to instructions are writing. Now, the ordering change the finally, what value will flow will be different that is why, instead of the final return value corresponds to  $j$  instead of  $i$ . So, that can happen are the program flow will decide one particular order and outcome of result is dependent on that, but the execution order may be different and such case they will be or the in the result will not be correct and that is known as output dependences and name dependence are not true dependence.

So, why we are calling name dependence, because register or memory locations are essentially considered as name in which when we are storing operands, how can overcome this, this output dependence not really true data dependence and they can be very easily overcome by technique known as a resistor relevant. So, later on I shall discuss more details, you can modify the name of the resistors, we can rename the resistor and this problem will be. So, the output dependences can be overcome and then we can change the order of the execution.

(Refer Slide Time: 43:04)

**Control Dependences**

- Determines the ordering of an instruction with respect to a branch instruction. Example:

```
if p1 {  
    S1;  
};  
if p2 {  
    S2;  
};
```
- S1 is control dependent on p1, but S2 is not control dependent on p1
- An instruction that is control dependent on a branch cannot be moved **before** the branch
- An instruction that is not control dependent on a branch cannot be moved **after** the branch

Ajit Pal, IIT Kharagpur

Finally the third type of dependences is known as control dependence, so determines the ordering of one of instructions with respect of the branch instructions. For example, here, if p is true p 1 is true, then S 1 and we have got another if than an a statement is p 2 is true then S 2. Now, whenever we have got this type of if then type of statement present in your program, we call S 1 is control dependent on p 1, because if p 1 is true only then you are doing this, then S 1 something else we will do.

And similarly S 2 is control dependent on p 2, now whenever this kind of control dependents are present, an instructions that is control dependent on a branch cannot be move before the branch. The means an instruction that is present in S 1, cannot be move before p 1, because then you may not get correct result, if you moves some instructions from that, then part of the program before that this is instruction p 1. So, this would not be done, similarly instruction that is not controlled dependent on a branch cannot be moved after the branch.

That means, some instruction which is present before, if p i should not be moved to this part that this than part, so that also should not be done, because in such a case that particular instructions will become control dependent on the p 1, which was not true earlier. That means, so whenever you do some kind of reordering instructions, you have to consider these dependences and you should not modified programs in such way.

(Refer Slide Time: 45:17)

**Control Dependences**

- When program order is strictly preserved, it ensures that control dependences are also preserved
- Example:

DADDU	R2,R3,R4
BEQZ	R2,L1
LW	R1,0(R2)

L1:

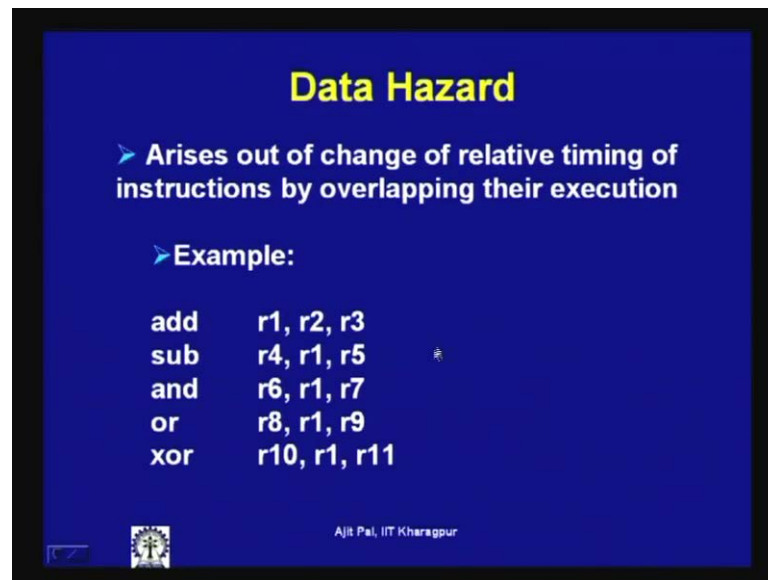
Ajit Pal, IIT Kharagpur

So, when program order is strictly preserved, it ensures that control dependent are also preserved. So, essentially requirement is to maintain the program order and if it is taking for the preserved, than control dependence are also preserved for example, a simple example has been taken here, say the first two instructions at data dependent, so they cannot be reorder, but you can see, so here it is producing some value, which is stored in R 2 that is being used in the second instructions, branch instruction, so you have got data dependence.

So, this data dependence can be very easily detected and obviously, while executing you are program that program order will be maintained, but you can see second and third instructions BEQZ or 2, L 1 and here you have got load R 1, 0 R 2. Here you can see there is no as such that, data dependence there is no flow of data between these two instructions. However, you cannot really reorder these two instructions, because if you do the reordering of this instructions, then this may lead to incorrect results, because whenever you are executing in program in a dynamic situation, in the input to an instruction make from multiple sources.

There is a possibility of coming the input from the multiple sources and that is dynamically dependent on the execution and when you execute a program. So, this particular although there is go data dependence, but if you try to reorder these two instructions it will lead to problem.

(Refer Slide Time: 47:15)



**Data Hazard**

- Arises out of change of relative timing of instructions by overlapping their execution
- Example:

```
add    r1, r2, r3
sub    r4, r1, r5
and    r6, r1, r7
or     r8, r1, r9
xor    r10, r1, r11
```

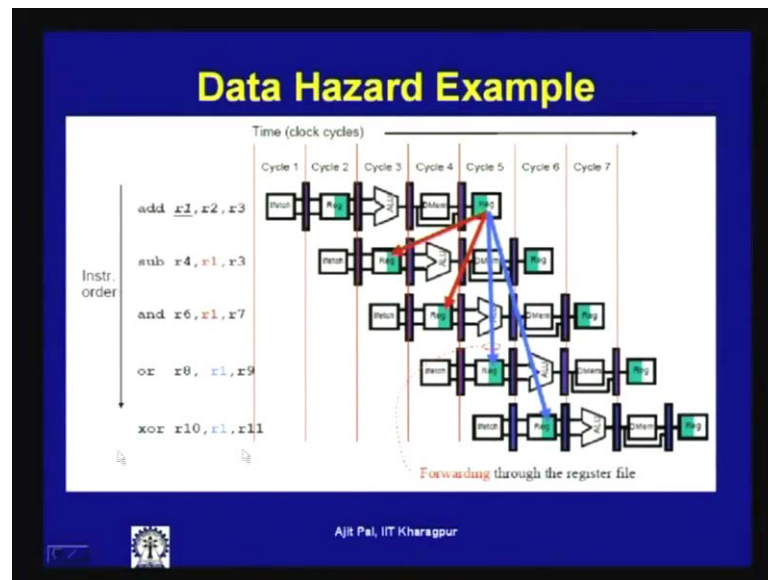
Ajit Pal, IIT Kharagpur

Now, I have already discussed about the data hazards in detail, it arises out of change of relating timing of instructions by overlapping their executions. So, here we have got a simple program and there is lot of data dependence as you can see, that R 1 the result is being stored in a R 1 in the first instruction, which is used as a source of one of the operands in the second instructions, that is also used as source operands in third instructions.

Now, a chain of instructions is several instructions one of at the other is data dependent on the first instructions. Now, how long this that are dependence you will persist, whenever you go for pipeline execution, actually it depends on the pipeline depth and it data dependence does not always mean, that it will lead to hazards.



(Refer Slide Time: 48:24)



So, let us see this particular diagram, this will help you to understand how and when it happened, so you can see here the four instructions following this first instruction `add r1, r2, r3` you have got 3 is to 4 instructions `sub r4, r1, r3` and `r6, r1, r7` or `r8, r1, r9` then `xor r10, r1, r11`. So, we find all this subsequent four instructions are data dependent on the first instruction, but we all of them lead to data hazards and stalls, so you can see here writing of the result is taking place in the 5th cycle.

If you look at the execution of the first instruction, instruction fetch, instruction decode, execution then the memory operation which is not present in the sense instructions, but cycle in be necessary. So, in 5th cycle you are writing the result in to the register, we find that before we 5th cycle was reached, in the 3rd cycle itself, second instruction has already read the data. So, there is a red line and which implies that, here there is a data hazard, similarly in this case also before the writing of the value in the register took place, the reading of the operand took place in the third instruction.

So, as a consequence you can see, there will be a hazard for the second instruction as well, but so far as the third and fourth instruction are concerned, they will not be, it will not lead to a hazard. Because, what has been done a technique known as split cycle has been used, where writing is done in the first phase of the clock, so you have clock as got two phases ((Refer Time: 50:44)). So, you are performing writing in this phase and reading your performing in the second phase, so this is a particular clock cycle.

So, whenever you do this split page, then only the third instruction will not suffer from hazard is split cycle is not done, then the third instruction also will suffer data hazards. So, in this particular case it is not red, because you are able to read the register value after it has been written. So, we find that although there are four instruction consecutive an instruction or data dependent on the first instruction, but only the first two is leading to data hazards, the next two there is no data hazards.

(Refer Slide Time: 51:44)

### Classification of Data Hazards

- Let I be an earlier instruction, J a later one
- **RAW** (read after write)
  - J tries to read a value before I writes into it
- **WAW** (write after write)
  - I and J write to the same place, but in the wrong order
  - Only occurs if more than one pipeline stage can write (in-order)
- **WAR** (write after read)
  - J writes a new value to a location before I has read the old one
  - Only occurs if writes can happen before reads in pipeline (in-order)

I: add r1, r2, r3  
 J: sub r4, r1, r3

I: sub r1, r4, r3  
 J: add r1, r2, r3

I: sub r4, r1, r3  
 J: add r1, r2, r3

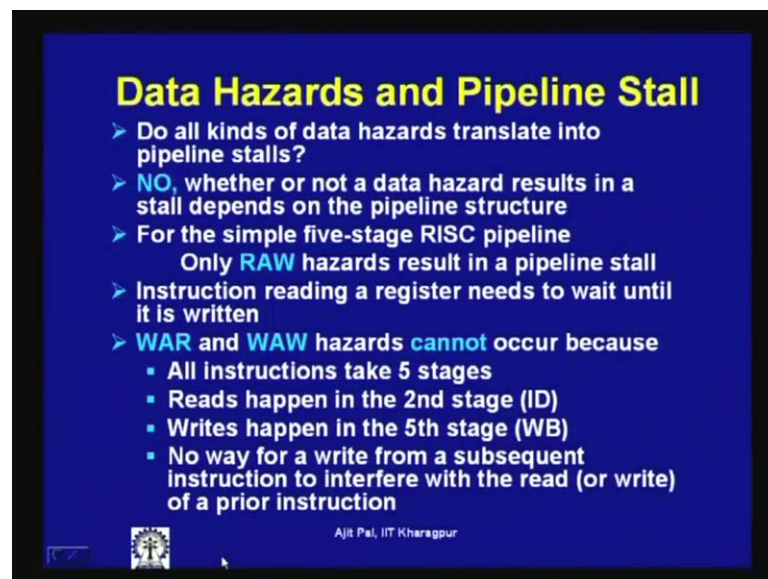
Ajit Pal, IIT Kharagpur

So, you can classify the data hazards with the help of three different ways, so classification of data hazards, first one say let us assume that, we have got an instruction I, be an earlier instruction J is a later one. Then read after write hazards is defined as J tries to read a value, before I write into it, which is known as read after write hazards as it shown here, this is the read after write hazard. You are writing some value in the register in the second instruction is reading the value, read after write, then the second one is to write after write, I and J write to the same place, but in the wrong order.

So, you can see if they are writing in the same place, but if they are order is different, then it will lead to an hazards known as write after write hazards, so only occurs in more than one pipeline stage can write. So, in our case that simple five stage pipeline that we have already discussed, were the second type of data hazards that is write after write type of the data hazards will not be present then think over it and will see this will not be present.

Then the third type of data hazards that is known as write after read, so in this case J writes a new value to your location before I as read the old value, so the J writes to new value before I has read it. So, in this also will not occur in our five stage pipeline, so this occurs only of when if writes can happen before reads, in our case we have seen that write operation take place at the last cycle. So, before that reading always take place, so this type of hazards will not be present in the five stage pipeline that we have already discussed.

(Refer Slide Time: 53:53)



**Data Hazards and Pipeline Stall**

- Do all kinds of data hazards translate into pipeline stalls?
- **NO**, whether or not a data hazard results in a stall depends on the pipeline structure
- For the simple five-stage RISC pipeline
  - Only **RAW** hazards result in a pipeline stall
- Instruction reading a register needs to wait until it is written
- **WAR** and **WAW** hazards **cannot** occur because
  - All instructions take 5 stages
  - Reads happen in the 2nd stage (ID)
  - Writes happen in the 5th stage (WB)
  - No way for a write from a subsequent instruction to interfere with the read (or write) of a prior instruction

Ajit Pal, IIT Kharagpur

So, we can conclude our lecture this statements, do all kinds of data hazards translate into pipeline stalls, we have the seen the answer is no, weather are not a data hazards results in a stall depends on the pipeline structure. And for the simple five stage risk pipeline that we have discussed only lead after write hazards result in a pipeline stall, and instruction reading a register needs to wait until it is written.

So, WAR and WAW hazards cannot occur, because all instructions takes five stages, reads happen in the second stage we have seen, write happens in the fifth stage, no way for a write from a subsequent instruction to interfere with the read of a prior instruction. So, we can conclude that for our simple five stage pipeline, only the first type of hazards, read after write hazards can occur, the other two types of hazards like a write after read and write after write type of hazards cannot occur.

But, there can be more complex pipeline, the pipeline depth may be much longer, latter on we shall discuss some examples. And there this type of hazards may also be present, all I am trying to tell for our simple pipeline, it does not second two types of hazards may not to lead to, I mean second two types of data dependence thus may not lead to hazards, does not mean they will not hazards at any point of time. They can lead to hazards, whenever your pipeline depth is longer or you have different types of pipeline structure, so with this let us stop here.

Thank you.