

**Computer Architecture and Organization**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 41**  
**Pipeline Scheduling**

We have seen so far what is a pipeline, how a pipeline works. In this lecture we shall be talking about Pipeline Scheduling, which means how we can feed the inputs to a pipeline such that no conflict or so called collision occurs in any one of these stages.

Now for a linear pipeline where the stages are connected as a linear chain this problem does not arise because we can potentially apply one new set of inputs every clock cycle. But if there are feed forward and feedback connections in a pipeline in general for a non-linear case, then the calculation of the schedule is not so easy or trivial.

So, here we shall be basically talking about Pipeline Scheduling.

(Refer Slide Time: 01:16)

**Scheduling of Non-linear Pipelines**

Two operations X and Y

- X: 8 time steps to complete
- Y: 6 time steps to complete

	1	2	3	4	5	6	7	8
S <sub>1</sub>	X					X		X
S <sub>2</sub>		X		X				
S <sub>3</sub>			X		X		X	

	1	2	3	4	5	6
S <sub>1</sub>	Y				Y	
S <sub>2</sub>			Y			
S <sub>3</sub>		Y		Y		Y

We take the same example of a non-linear pipeline that we saw earlier. This is a simple enough example that you can illustrate. There are three stages: S1 S2 S3. Here we are not showing the inter-stage latches that are also there, but this is only for illustration. This arrow shows how the data flows from one stage to the other. And here we are assuming that not only the pipeline is non-linear in the sense that there can be data flow from say

S3 to S1, S1 to S3 and so on, but it is a multifunction pipeline; which means, we have the same pipeline which can either compute a function X or compute a function Y.

You can take an analogy for an arithmetic pipeline where you can have a general pipeline where both multiplication and addition can be carried out. But of course, all of the stages will not be used for the two cases. We can selectively use some cases, some of the stages may be using for more than one clock cycles and so on.

So, the operation X that is represented by this reservation table, requires 8 cycles to complete and this is the sequence of stage occupancy. First the input data goes to S1, then to S2, then to S3, then again to S2; you see S3 to S2 there is a connection, from S2 again to S3, S3 to S1; you see S3 to S1 also there is a connection. Then S1 to S3 there is a feed forward connection like this and finally S3 again to S1 then it finishes. So, from S1 it finishes or it goes out.

Similarly for the function Y it again starts with S1 it goes to S3, S2, 3, S1, and finally S3 where it finishes; so from S3 it exits. So, X will exit from here, Y will exit from here. With the help of this example let us explain how scheduling of this pipeline can take place. This means, say the first input which during computation maybe in S2, the second input is also trying to go into S2 at the same time. So, it cannot happen that we know, so one of them has to wait.

(Refer Slide Time: 03:59)

• Latency Analysis:

- The number of time units between two initiations of a pipeline is called the *latency* between them.
- Any attempt by two or more initiations to use the same pipeline stage at the same time will cause a *collision*.
- The latencies that can cause collision are called *forbidden latencies*.
  - Distance between two X's in the same row of the reservation table.

	1	2	3	4	5	6	7	8
S <sub>1</sub>	X					X		X
S <sub>2</sub>		X		X				
S <sub>3</sub>			X		X		X	

Forbidden latencies: 2, 4, 5, 7

	1	2	3	4	5	6
S <sub>1</sub>	Y				Y	
S <sub>2</sub>						
S <sub>3</sub>		Y		Y		Y

Forbidden latencies: 2, 4

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL TECHNOL

We shall be doing something called latency analysis. Latency is actually the delay between two successive data inputs or initiations that are fed to the pipeline. Our objective is to calculate the latency value or the sequence of latency values that will not result in collision in any of the stages.

Latency is basically defined as the number of time units or clock cycles between two successive initiations; two inputs applied to the pipeline what is the minimum number of time steps we need to give between them. And as I had said if we are not careful enough then two or more initiations may try to use the same stage at the same time that results in a collision.

Some of the latencies will definitely result in a collision, and such latencies are referred to as forbidden latencies; we cannot use those latencies. How you can estimate the forbidden latencies? You can simply calculate the distance between two check marks in the rows of the reservation table. Let us take the example reservation table for X. You see here there are so many check marks, there are check marks at a gap of 2, there are check marks at a gap of 4, there are check marks at a gap of 5, and gap of 7.

These are all forbidden latencies; distance between any two pairs of check marks. For this reservation table for X, the forbidden latencies are 2, 4, 5, and 7. For the other function Y, here you see 2 and 4, here also it is 4. Here the forbidden latencies are only 2 and 4. This is the first step in our analysis to list or enumerate the forbidden latencies.

(Refer Slide Time: 06:33)

— A *latency sequence* is a sequence of permissible non-forbidden latencies between successive task initiations.

— A *latency cycle* is a latency sequence that repeats the same subsequence.

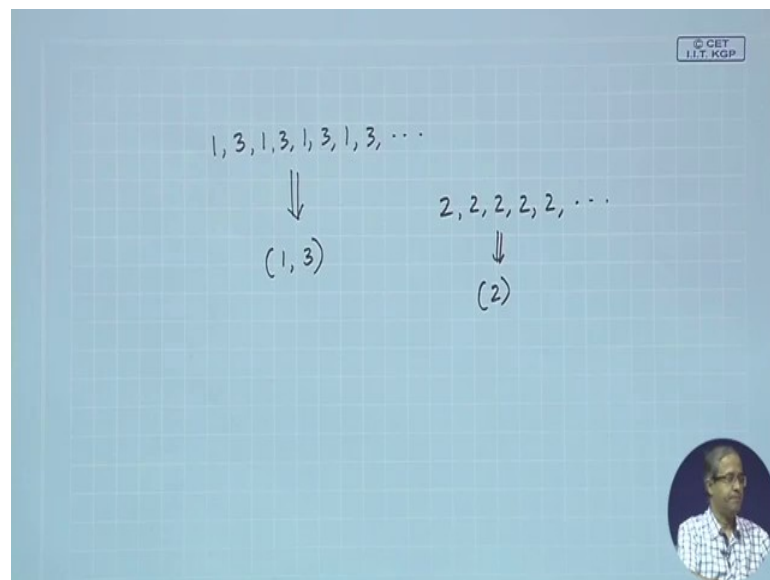
Function X	Function Y
<ul style="list-style-type: none"> <li>Forbidden latencies: 2, 4, 5, 7</li> <li>Possible latency cycles:               <ul style="list-style-type: none"> <li>(1, 8) = 1, 8, 1, 8, ... (average latency = 4.5)</li> <li>(3) = 3, 3, 3, ... (average latency = 3.0)</li> <li>(6) = 6, 6, 6, ... (average latency = 6.0)</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Forbidden latencies: 2, 4</li> <li>Possible latency cycles:               <ul style="list-style-type: none"> <li>(1, 5) = 1, 5, 1, 5, ... (average latency = 3.0)</li> <li>(3) = 3, 3, 3, ... (average latency = 3.0)</li> <li>(3, 5) = 3, 5, 3, 5, ... (average latency = 4.0)</li> </ul> </li> </ul>

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES
 NATIONAL INSTITUTE OF TECHNOLOGY

Now a latency sequence is defined as a sequence of permissible non-forbidden latencies that do not result in a collision. I can say that I am using a latency sequence of 1 2 1, which means between the first and second input I give a gap of 1, between second and third input I give a gap of 2, between third and fourth input I again give a gap of 1.

This sequence can have any arbitrary delay gaps that you can specify. And if this sequence repeats in time then we call it is a latency cycle. Latency cycle is nothing but a latency sequence where the same subsequence is repeated.

(Refer Slide Time: 07:44)



Let us take an example: suppose we use a latency sequence like this: 1, 3, 1, 3, 1, 3, and this repeats indefinitely. We can write in a compact notation that it is a latency cycle (1, 3); this 1, 3 will be repeating indefinitely. Similarly if I have a case for the same latency 2, 2, 2, 2, 2 is to be used, then I can write it as a latency cycle containing a single latency (2).

Talking about the two function: the first function X we have already seen that the forbidden latencies are 2, 4, 5, and 7. Here we are just listing some of the forbidden latency cycles or the some of the latency cycle that you can use, but how to get this we shall be showing shortly. Let us say one of the possible latency cycles that do not result in collisions is (1, 8). So, 1, 8, 1, 8 like that you proceed. This will result in an average latency of 4.5, because (1 + 8) is 9 divided by 2.

If it is a 3, 3, 3, like this here average latency is 3 because 3 is a non forbidden latency and multiples of 3's are also not there. Similarly, (6) is also a feasible latency cycle 6, 6, 6, with average of 6. In this case the minimum latency is resulting for the second case.

Similarly, for the function Y where the forbidden latencies are 2 and 5 we have several alternatives. We can use 1, 5, 1, 5, this kind of a cycle, because 1 is not forbidden, 5 is also not forbidden; this results in an average of 3. We can use 3, 3, 3, this also results in average of 3, also you can use 3, 5, 3, 5 alternating; for this average will be 4.

Now the cycles where there is no change, the same latency have to be used, are referred to as constant cycles. Like here (3), (6) and here (3), these are examples of constant cycles.

(Refer Slide Time: 10:04)

**Collision Free Scheduling**

- Main objective:
  - Obtain the shortest average latency between initiations without causing collisions.
- We define a *collision vector*.
  - If the reservation table has  $n$  columns, the maximum forbidden latency is  $m \leq n-1$ .
  - The permissible latencies  $p$  will satisfy:  $1 \leq p \leq m-1$ .
  - The collision vector is an  $m$ -bit binary vector  $C = (C_m C_{m-1} \dots C_2 C_1)$ , where  $C_i = 1$  if latency  $i$  causes collision, and  $C_i = 0$  otherwise.
  - $C_m$  is always 1.

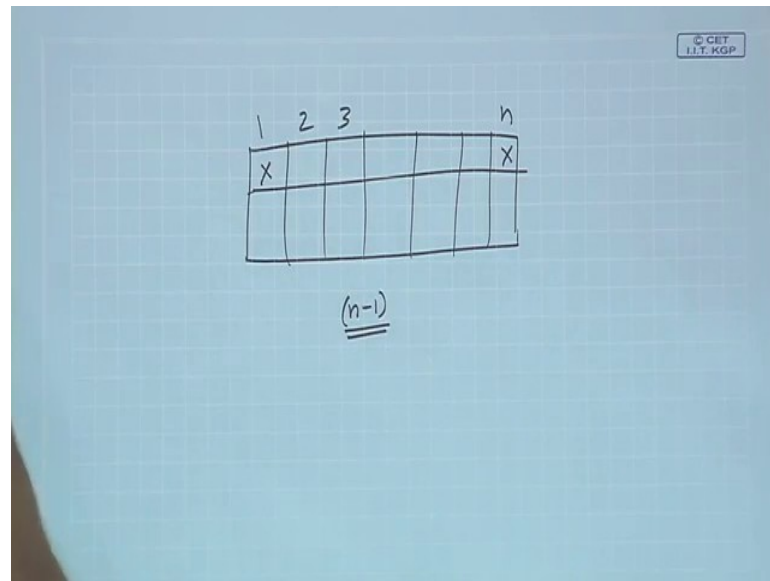
Function X:  $C_x = (1011010)$   
 Function Y:  $C_y = (1010)$

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES
 NATIONAL INSTITUTE OF TECHNOLOGY

Now, our objective is to do something called collision free scheduling. How to schedule the pipeline, how to determine the latencies dynamically? Dynamically means when the data comes should be able to automatically find out whether we can feed the next data or not; that is called scheduling. We shall be looking into this.

Scheduling such that there is no collision; well of course, the main objective is to obtain the shortest average latency. Well we start by defining a data structure or vector you can call collision vector. It is a bit vector. Collision vector is defined as follows: if there are  $n$  numbers of columns in the reservation table then it is like this.

(Refer Slide Time: 11:04)



If you have a reservation table there are rows, there are many columns, let us say there are  $n$  number of columns. In the worst case there can be a check mark here and check mark here that will result in a forbidden latency of  $n - 1$ , this can be the maximum forbidden latency.

Here we have mentioned the same thing; the maximum forbidden latency  $m$  cannot be greater than  $n - 1$ . And the permissible latencies that we shall be using for scheduling this obviously has to be less than or equal to  $m - 1$ , because more than  $m$  does not make sense, because effectively if there are 5 stage in the pipeline and I am saying that I have to give a gap of 6 then there is no use in having the pipeline. Anyway, I will be waiting for 6 time cycles to feed the next data. So, that is not a good way of scheduling. We will say that the permissible latencies  $p$  will be satisfying this inequality, maximum forbidden latency is  $m - 1$ , and minimum is of course 1.

The collision vector is an  $m$ -bit vector, where  $m$  is this maximum forbidden latency,  $C_1, C_2$  to  $C_m$  where a particular bit is 1 if latency  $i$  causes a collision, and that bit is 0 if the latency  $i$  does not cause a collision. And because  $m$  is the maximum forbidden latency by definition  $C_m$  is always 1, because  $m$  is the maximum forbidden latency.

So, for the function  $C_x$ , the forbidden latencies were 2, 4, 5, and 7. The collision vector is shown. And function  $Y$  from the right side again 2 and 4, these are the forbidden latencies. So, the collision vectors for the two cases are defined like this.

(Refer Slide Time: 13:23)

• From the collision vector (CV), we can construct a *state diagram* specifying the permissible state transitions among successive initiations.

- The collision vector corresponds to the initial state of the pipeline.
- The next state at time  $t+p$  is obtained by shifting the present state  $p$ -bits to the right and OR-ing with the initial collision vector  $C$ .

Function X  
CV: (1011010)

Function Y  
CV: (1010)

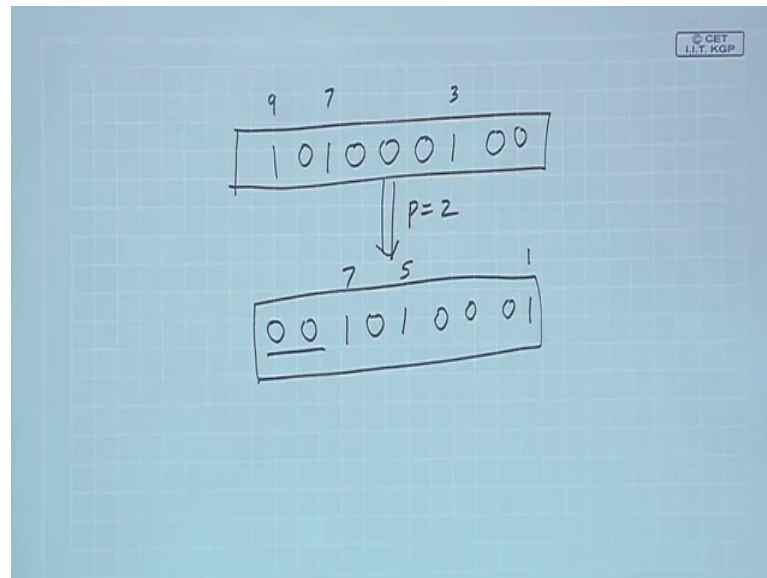
The slide contains two state diagrams. The left diagram for Function X starts at state 1011010. Transitions are labeled with time gaps: 8+ (self-loop), 3 (to 1011011), 6 (to 1011011), 8+ (to 1111111), 1 (to 1111111), and 8+ (to 1111111). States 1011011 and 1111111 have self-loops labeled 3 and 6 respectively. The right diagram for Function Y starts at state 1010. Transitions are labeled with time gaps: 5+ (self-loop), 3 (to 1011), 5+ (to 1011), 1 (to 1111), and 5+ (to 1111). State 1011 has a self-loop labeled 3.

Now from the collision vector we construct a state diagram where we define the states as the status of collisions at a particular point in time. That means which time gaps or latencies can result in a collision at that point in time; that is one state. And the state transition diagram can contain multiple states; you can go from one state to the other.

And the initial state is the collision vector. And from the collision vector from any state you can move to some other state by advancing the time by some number  $p$ . Suppose you are currently in time  $t$ . If you advance the time by  $p$ , what you do. Whatever is the present state you shift it right by  $p$  positions.



(Refer Slide Time: 14:31)



See what is the meaning? Suppose I am in a present state where my bits are something like this, which means that from time 1, 2, 3, 4, 5, 6, 7, 8, and 9; 3, 7, and 9 these are my forbidden times where I cannot feed the next input --- these are forbidden latencies. But now say if I advance my time by 2 steps then whatever was 3 now after two steps this will become 1, 7 will become 5, 9 will become 7. So, actually we are shifting this right by 2 positions. So, this 1 will come here this 0 0 1 0 1 and two zeros will be shifted in. This will be my new status after time 2. Now the collisions will occur at time 1, 2, 3, 4, 5, and 7; this is the idea.

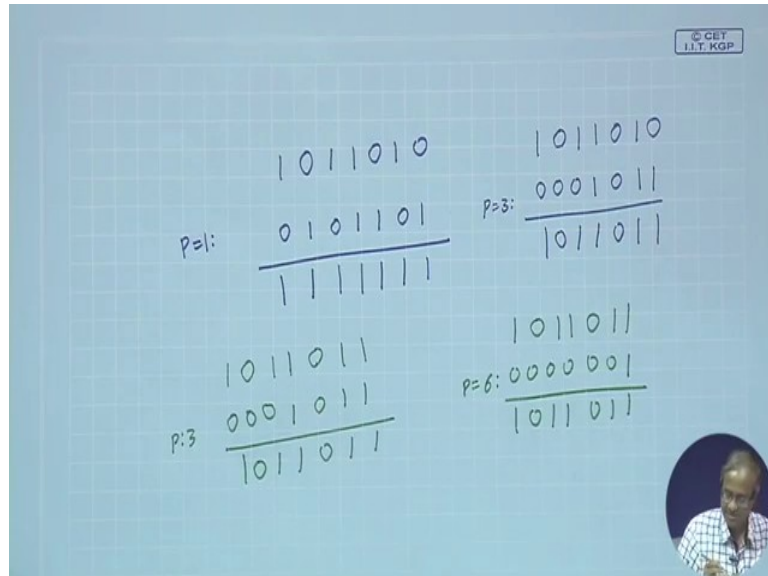
So, the next state is obtained by shifting it right, and of course the initial collision vector that was there that lists the forbidden latencies. The latencies that are forbidden will always remain forbidden. So, whatever you have obtained by right shifting you will have to do a bit by bit logical OR with the initial collision vector, so that whenever there is a 1 in the initial collision vector it will remain a 1 in all the states, because those will always be forbidden you cannot use that latency at any point in time.

Let us take that same example. This is the state diagram for function X. Let us see how we have obtained this. We have said that for this function the collision vector was this; it indicates the forbidden latencies of 2, 4, 5, and 7. So, this will be my initial state. Now from my initial state I can advance my time only by that amount where I have 0's that are



not forbidden, which means time 1, 3, 6 or 8, or more. If I advance it by 1 you see my collision vector was 1 0 1 1 0 1 0.

(Refer Slide Time: 17:04)



If I advance it by time 1, I will be doing a right shift 1 0 1 1 0 1 and 0 will come in; then you do a bit by bit OR.

So, from this initial state that represents the collision vector if I go or advance time by 1, I go to this state just like I calculate it. If I advance it by 3, I go to this state, if I advance by 6, this you can check you will also arrive at the same vector.

But if I advance it by 8; that means, if you shift it by 8 everything will become 0, 0 OR with the initial vector will be the initial vector itself. Similarly from here there are no zeros, so you can only have 8 or more; shift it by 8 positions all will be zeros or with this you get back this. So, this will be an arrow back. And so on.

(Refer Slide Time: 20:47)

From the state diagram, we can determine latency cycles that result in *minimum average latency (MAL)*.

- In a *simple cycle*, a state appears only once.
- Some of the simple cycles are *greedy cycles*, which are formed only using outgoing edges with minimum latencies.

<b>Function X:</b> <ul style="list-style-type: none"> <li>Simple cycles: (3), (6), (8), (1,8), (3,8), (6,8)</li> <li>Greedy cycles: (3), (1,8)</li> </ul>	<b>Function Y:</b> <ul style="list-style-type: none"> <li>Simple cycles: (3), (5), (1,5), (3,5)</li> <li>Greedy cycles: (3), (1,5)</li> </ul>
---	---

**MAL = 3 for both X and Y**

From the state diagram we can determine the latency cycles such that the minimum average latency is minimized. When we talk about a simple cycle a state appears only once. This is of course the kind of cycle that you want, because we do not want a cycle like this. for I am telling 3, 3, 8, 3, 3, 8, --- 3 is appearing twice. I shall only be considering simple cycles. And greedy cycles are those where from every state we shall only be taking the outgoing edges with minimum label.

(Refer Slide Time: 22:33)

### The Scheduling Algorithm

```

Load collision vector (CV) in a shift register R;
If (LSB of R is 1) then
begin
  Do not initiate an operation;
  Shift R right by one position with 0 insertion;
end
else
begin
  Initiate an operation;
  Shift R right by one position with 0 insertion;
  R = R OR CVorig; // Logical OR with original CV
end
  
```

Now, scheduling algorithm goes as follows. Suppose I want to build a control circuit that will automatically tell me whether it is safe to feed the next data or not. We load collision vector in a shift register. This is my original collision vector, and I load my collision vector in the shift register initially.

At every point in time I do a right shift, where I feed a 0 and I check whether is 0 is coming out or a 1 is coming out. If a 1 is coming out it means that next time instant is forbidden, you cannot feed a data at that time. If the last bit that is coming out is 1 then do not initiate an operation simply shift the register by 1 bit with 0 insertion, and in the next iteration again check whether you are allowed to feed or not.

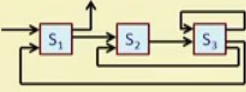
But if you find that it is a 0 what does that mean with respect to the state diagram; that here 0 means you are going to the next state. Next state means you have to shift right and OR with the collision vector to get the new state. The same thing we are doing here. Here you see: if the bit is 0 initiate an operation then we shift it right and then we OR-ed with the collision vector; whatever is the shifted value we OR-ed with the collision vector and load back into the shift register, and this process repeats.

If you just think, whatever we are doing here is exactly the process by which we were computing the state diagram. At every step we are trying to shift it by a number of bits that is allowed. We go on shifting till the next 0 comes, then we OR it with the collision vector. And we always move when you get the first 0; that means we are looking only for greedy cycles. From every state we are moving out following the smallest or the least value of the cycle.

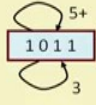
(Refer Slide Time: 25:16)

### Optimizing a Pipeline Schedule

- We can insert non-compute (dummy) delay stages into the original pipeline.
  - This will modify the reservation table, resulting in a new collision vector.
  - Possibly a shorter MAL.



	1	2	3	4	5
S <sub>1</sub>	X				X
S <sub>2</sub>		X		X	
S <sub>3</sub>			X	X	



**MAL = 3**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, there are ways to optimize a pipeline schedule. I am showing you one method with the help of an example. Suppose we have a non-linear pipeline like this where the reservation table is as follows. You see in some of the time steps like 4, two of the stages are simultaneously used. And here the trial forbidden latencies are 1 2 and 4; so 1 0 1 1 is the CV. The state diagram will be very simple; you can either shifted by 3 or 5 or more; obviously, 3 is the minimum average latency.

Now what you are exploring here is that: can we insert some delay elements in the pipeline, some dummy stages which do not compute anything just it take up some clock cycles? You may say that well if you insert dummy stages that do nothing simply eat up some cycles, then my time will be increasing. Well your time maybe increasing for a single computation, but when you are feeding many data one after the other earlier there were lot of collisions, but if you insert those delays maybe the number of collisions will become less; that is the idea.

(Refer Slide Time: 26:42)

• Suppose we insert delay elements  $D_1$  and  $D_2$  as shown.

– This will modify the reservation table, resulting in a new collision vector.

	1	2	3	4	5	6	7
$S_1$	X					$\rightarrow$	$\rightarrow$ $X_2$
$S_2$		X		X			
$S_3$			X		$\rightarrow$ $X_1$		
$D_1$				X			
$D_2$							X

MAL = 2 (for the greedy cycle (1,3))

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So for this example, suppose I insert delays like this. This was my original pipeline; I insert delays in two of the branches: one delay here one delay here. That means, while going from  $S_3$  to  $S_1$  I put a delay, and  $S_3$  to  $S_3$  also I put a delay. So, earlier my reservation table was this, now in the reservation table there will be two additional dummy stages introduced. And because of the delay some of the check marks will get shifted: this will get shifted here, this will get shifted here. And see here after  $S_3$  you first go to  $D_1$ , then you are going back to  $S_3$ . Earlier you are directly going to  $S_3$ . Similarly for this from  $S_3$  you are going to  $D_2$ , from  $D_2$  then you are going to  $S_1$ . Now for this one again it is a slightly more complex here the forbidden latencies are 2 and 6.

If you just compute the state transition diagram in a similar way you can check this. This is the state transition diagram that is obtained. And if you compute the greedy cycles in this case this will be the best cycle (1, 3); 1 3 1 3 1 3 that will be result in an average of 2. Earlier it was 3 now by inserting the delays we have achieved an average of 2. So, by inserting this kind of dummy delay stages the performance of a non-linear pipeline can be improved, this is something we have to keep in mind.

(Refer Slide Time: 28:34)

**Exercise 1**

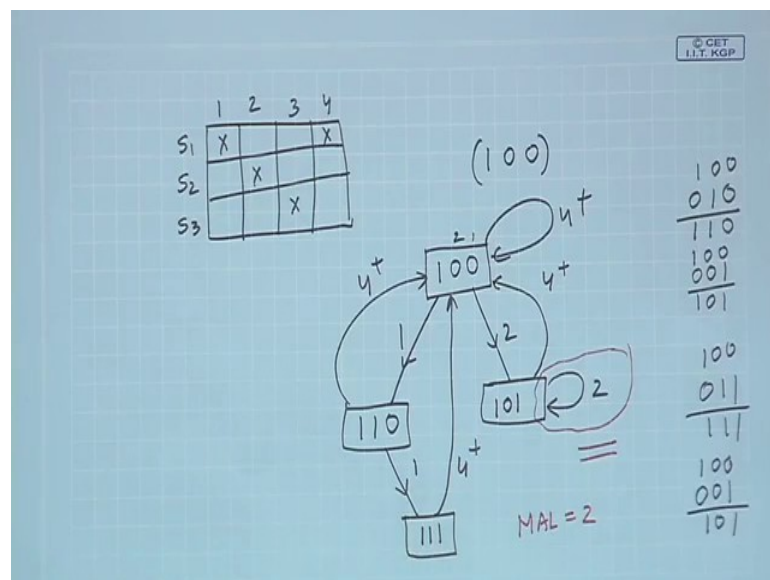
- For the following reservation tables,
  - What are the forbidden latencies?
  - Show the state transition diagram.
  - List all the simple cycles and greedy cycles.
  - Determine the optimal constant latency cycle, and the MAL.
  - Determine the pipeline throughput, for  $\tau = 20$  ns.

	1	2	3	4
S <sub>1</sub>	X			X
S <sub>2</sub>		X		
S <sub>3</sub>			X	

	1	2	3	4	5	6	7
S <sub>1</sub>	X		X				X
S <sub>2</sub>		X			X		
S <sub>3</sub>			X		X		

Now, let us work out couple of examples. Let me work out one of them, and one of them I leave as an exercise for you. Let us take the smaller one.

(Refer Slide Time: 28:48)



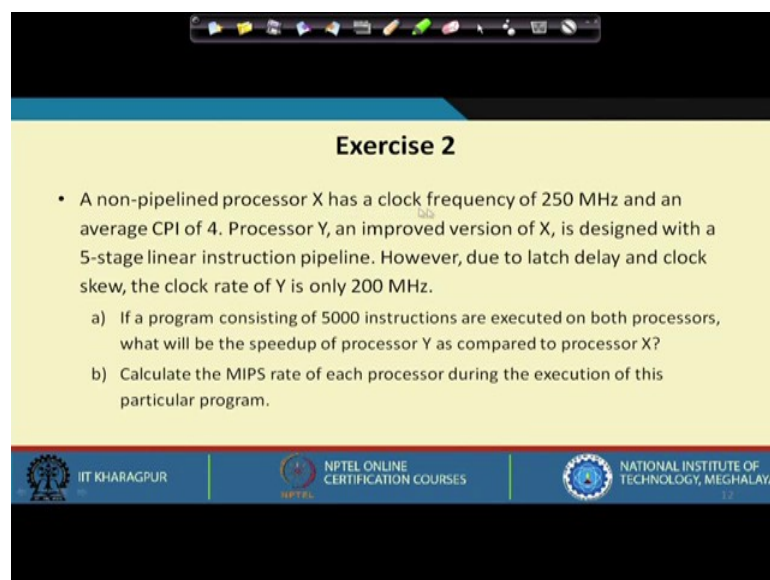
Here we have a reservation table, where there are three stages: S<sub>1</sub>, S<sub>2</sub> and S<sub>3</sub>; and there are four time steps which are required for a computation. This is a very simple example where my only forbidden latency is 3. So, my collision vector will (1 0 0). If I draw the state transition diagram this will be my initial state. Now I can shift it by either 1 or 2. If I shift it by 1 what will happen; I will shift it right by 1 position it will be 0 1 0, I OR it

with the original collision vector I get 1 1 0. So, my new state will be 1 1 0. If I shift it by 2 positions it will be 0 0 1, again if I OR it with the original collision vector I get 1 0 1.

Now once you are here the only 0 is here, if we shift it by 1 position it will become 0 1 1 and OR it with original CV it will become 1 1 1. So, if you shift it by 1 you land up in another state 1 1 1. And here 2 is only forbidden. If you shift by 2 positions it will be 0 0 1, OR with 1 0 0; it will remain in 1 0 1. So, if you have 2 it will remain here.

If we analyze this state transition diagram we see that the minimum cycle is (2). So, minimum average latency will be 2. So, like this given any reservation table you can first calculate your state transition diagram, then from here you have to find the cycle that results in minimum average latency; that will be your minimum.

(Refer Slide Time: 31:38)



**Exercise 2**

- A non-pipelined processor X has a clock frequency of 250 MHz and an average CPI of 4. Processor Y, an improved version of X, is designed with a 5-stage linear instruction pipeline. However, due to latch delay and clock skew, the clock rate of Y is only 200 MHz.
  - a) If a program consisting of 5000 instructions are executed on both processors, what will be the speedup of processor Y as compared to processor X?
  - b) Calculate the MIPS rate of each processor during the execution of this particular program.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us work out another example. This directly do not relate to instruction scheduling, but a problem of pipelining. Let us see what this states. It says a non-pipelined processor X where there is no pipelining has a clock frequency of 250 MHz and a CPI of 4. Because there is no pipelining it takes 4 clock cycles to execute one instruction. There is an improved version Y that is having a 5-stage linear pipeline, but because of the latch delays and other overheads the clock frequencies reduce to 200 MHz.



Now, two things are asked: that if I am running 5000 instructions on both the processors what will be the speedup. How fast will Y be with respect to X? And secondly, what will be the MIPS rating of the two processors?

(Refer Slide Time: 32:47)

© CET  
I.I.T. KGP

$$X: \quad XT = IC \times CPI \times CCT$$

$$XT_x = 5000 \times 4 \times \frac{1}{250M} \text{ Msec} = 80 \text{ Msec}$$

$$XT_y = 5000 \times 1 \times \frac{1}{200M} \text{ Msec} = 25 \text{ Msec}$$

$$\therefore \text{Speedup} = \frac{XT_x}{XT_y} = \frac{80}{25} = 3.2$$
  

X:	80 Msec	_____	5000 instrs.	
	$\therefore$ 1 sec	_____	$\frac{5000}{80}$ MIPS	= 62.5 MIPS
				↓
Y:	25 Msec	_____	5000 instrs.	
	$\therefore$ 1 sec	_____	$\frac{5000}{25}$ MIPS	= 200 MIPS

Let us work out for processor X. Let us calculate the execution time. We recall execution time is given by the instruction count multiplied by the cycles per instruction multiplied by the clock cycle time. So, for the first processor execution time comes to 80 microseconds.

Similarly, the execution time of the pipelined processor comes to 25 microseconds.

So, the speed up will be  $80 / 25 = 3.2$ .

Now, second part of the problem concerns the MIPS rating. Again for the machine X you see for executing 5000 instruction you require 80 microseconds. You can say that in 80 microseconds you are able to execute 5000 instructions. Therefore, in 1 second how many you execute? This comes to 62.5 million instructions per second.

Similarly, for Y it comes to 200 MIPS.

With this we come to the end of this lecture. In this lecture we talked about the various ways in which we can calculate the permissible latencies in a general pipeline. You see,

normally if it is just a linear pipeline all these calculations will not come into the picture. But if it is a more complicated kind of a pipeline with non-linear feed forward and feedback connections, then this latency calculation and latency analysis becomes important.

Thank you.