Digital Computer Organization

Prof. P. K. Biswas

Department of Electronics and Electrical Communication Engineering

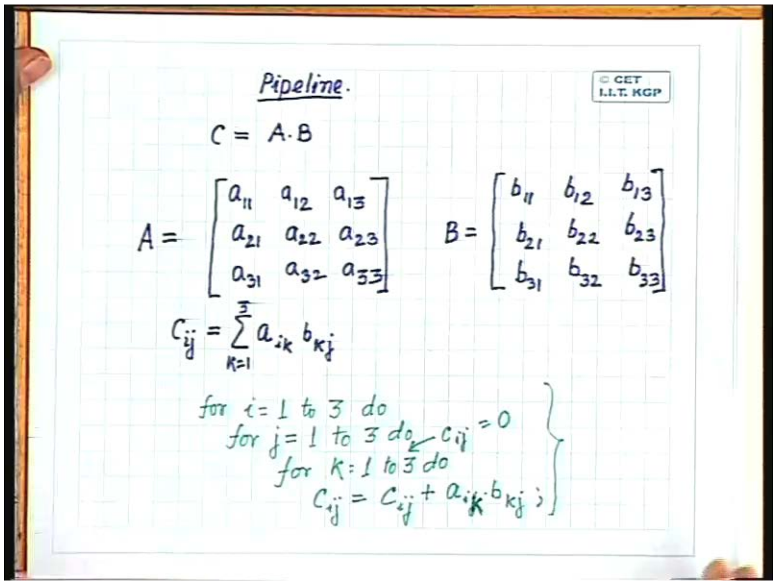
Indian Institute of Technology, Kharagpur

Lecture No. # 08

Pipeline Concept – II

Today let us consider designing of another pipeline architecture. You know that if we want to multiply two matrices, let us assume that we have two matrices A and B and we want to multiply the two matrices to generate a third matrix C.

Refer Slide Time: 00:01:17 min)



Pipeline.

© CET I.I.T. KGP

$C = A \cdot B$

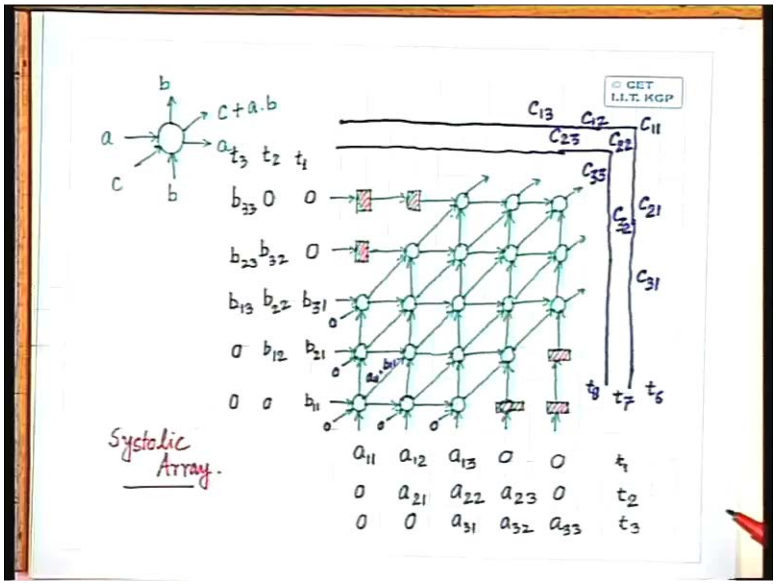
$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{3} a_{ik} b_{kj}$$

```
for i = 1 to 3 do
   for j = 1 to 3 do   c_ij = 0
      for k = 1 to 3 do
         c_ij = c_ij + a_ik · b_kj ;
```

The matrix multiplication is multiplying two matrices A and B to generate the third matrix C. We assume that this matrices A and B both are of dimension three. That is they have three number of rows and three number of columns. So for matrices A the elements are a_{11} a_{12} a_{13} a_{21} a_{22} a_{23} a_{31} a_{32} a_{33}. This is matrix A. Similarly matrix B is also of dimension three given by b_{11} b_{12} b_{13} b_{21} b_{22} b_{23} b_{31} b_{32} and b_{33} and you all know that if we want to multiply these 2 matrices, our resultant matrix will be given by, every ijth element in matrix C is given by a_{ik} multiplied with b_{kj} sum of k equal to 1 2 3. If we write a program for creating this matrix C, the program will simply look like this, that for i equal to 1 to 3 do, for j, a_{ik}. So it is c_{ij} equal to c_{ij} plus a_{ik} into b_{kj} and before this, at this point I have to initialize c_{ij} to zero. So I need such a for loop to compute this matrix C and you find that here, the complexity of this matrix multiplication root in will be 3 into 3 into 3, that is 27 number of operations. Now let us say whether we can design a pipeline architecture so that I can multiply this matrix in less amount of time.

(Refer Slide Time: 00:05:08 min)


Systolic Array.
c + a.b
t_3 t_2 t_1
b_{33} 0 0
b_{23} b_{32} 0
b_{13} b_{22} b_{31}
0 b_{12} b_{21}
0 0 b_{11}
a_{11} a_{12} a_{13} 0 0 t_1
0 a_{21} a_{22} a_{23} 0 t_2
0 0 a_{31} a_{32} a_{33} t_3
c_{13} c_{12} c_{11} c_{23} c_{22} c_{33} c_{32} c_{21} c_{31}
t_8 t_7 t_6
© CET I.I.T. KGP

So for this pipeline architecture I assume that every element in the pipeline architecture is having a function like this. It has got three inputs, inputs a b and c and it provides three outputs. In the horizontal and vertical directions whatever is the input, the same thing comes as output. So here the output will be a, here the output will be b, whereas in the diagonal direction the output will be c plus a into b. So this is the processing element using which we will try to implement, try to design a pipelines architecture so that I can multiply this 3 by 3 matrices. So the architecture will be something like this. So as we have said that every such processing element will have three inputs, one we represent as horizontal input, the other one as vertical input and we have a diagonal input. So it is like this, the inter connection will be met of this form.

So this is the type of inter connection that we will have and you will feed the data on this input sites. For feeding the data we will assume that during time t_1, the data that we feed to this vertical input or like this 0 0 a_{13} a_{12} and a_{11}. So that is one row of matrix a, during the same thing t_1 we have to feed in one column of matrix b to this horizontal inputs. So those are b_{11} b_{21} b_{31} and these inputs are zero. During time t_2 I feed the second row of matrix a in this vertical input. So that will be 0 a_{21} a_{22} a_{23} and this input is 0 and during same time t_2, I will feed the second column of matrix b to this horizontal input. So this will be 0 b_{12} b_{22} b_{32} and this input will be zero.

Similarly during time t_3 I feed the third row of matrix a to this vertical input and this becomes a_{31} a_{32} a_{33} and during the same time t_3, I feed in the third column of matrix b to this horizontal input. So this input becomes 0 0 b_{13} b_{23} and b_{33} and these are actually delay elements which are shown by rectangles or latches. Now given this type of diagram, let us see how this architecture works. You find that since each of these processing elements are of this type, that means whenever you feed in a as horizontal input, b as vertical input and c as the diagonal input, this will output a in the horizontal direction. The same thing is passed out, it will output b in the vertical direction whereas in the diagonal direction it will give c plus a into b.

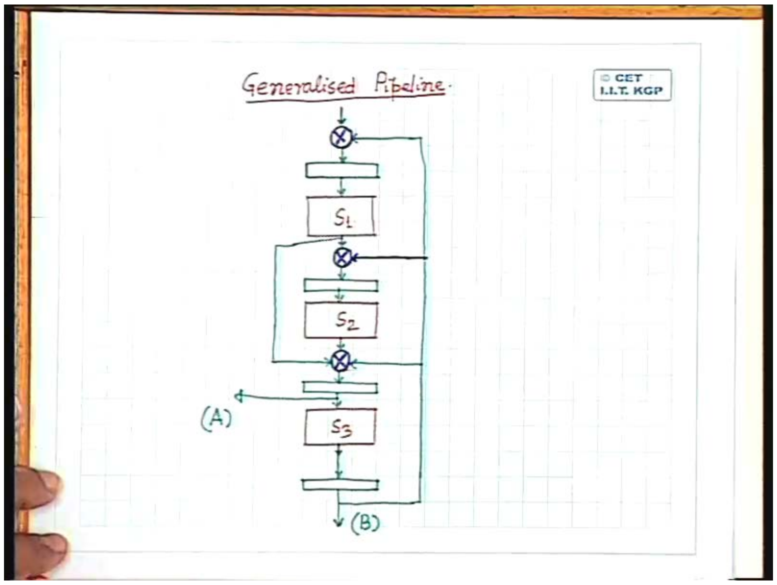
So if we assume this, you find that… in addition to this diagonal inputs are always zero, these are always zero. So you find that since at time t_1, we are feeding one row of matrix a in the vertical direction and feeding one column of matrix b in the horizontal direction. So once this data are available, what is the output of this processing node? That is 0 plus a_{11} into b_{11}. So at this point I will have the output as a_{11} into b_{11}. What I will have here? Simply a_{11} and in this direction I will have simply b_{11}.

Similarly after same time t_1 what I will have in this direction? Simply a_{12} will be passed here in the vertical direction because this is the vertical input, b_{21} which is fed horizontally that will be passed here. So what I get during t_2 is during t_2 here I am feeding a_{21} but during t_2, a_{12} is already available. Similarly during t_2 this b_{21} is already available here. The next data that I am feeding is b_{12}. So after time instant t_2 what I will get here? Here I have a_{12} available here. I have b_{21} available and in this diagonal direction I have a_{11} into b_{11}. So in this diagonal output, I will have a_{11} into b_{11} plus a_{12} which is already available on this link into b_{21} which is already available on this link.

So here I will have a_{11} into b_{11} plus a_{12} into b_{21} available on this link after time t_2. Similarly after time t_3 what I will get here is a_{11} into b_{11} plus a_{12} into b_{21} plus a_{13} into b_{31} that will be available here. That is after time t_3 and simultaneously because this data is flowing through this pipeline, all other components will also be generated. So if you analyze this way, you will find that the output will be something like this. You will get starting from time t_6 onwards, so I will put it like this. After t_5 that is t_6, during time t_6 what I will get here? c_{11}; c_{11} will be outputted by this. That is after time t_5 that is during time t_6, I will get c_{11} here. At the same time, during time t_6, I will get c_{21} which will be outputted by this node in the diagonal direction. During the same time t_6, I will get c_{31} which will be outputted by this node.

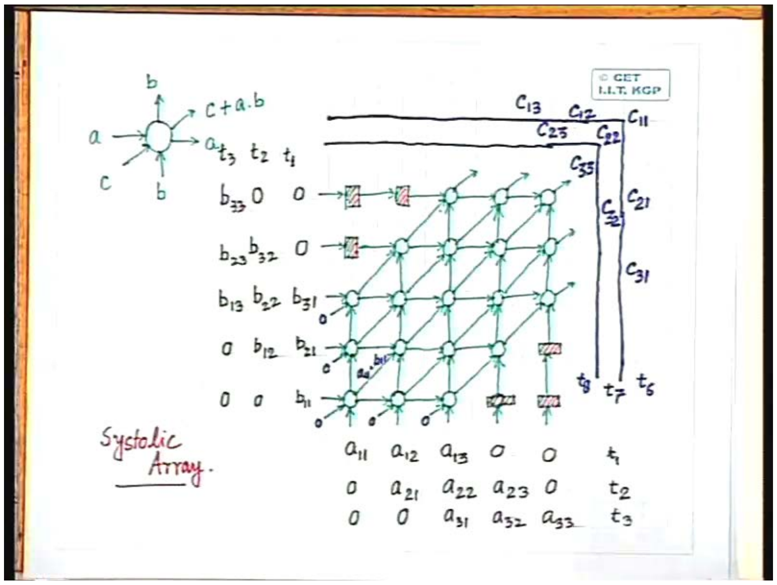
During same time t_6, this node will output c_{12} and this node will output c_{13}. During time t_7 this node will output c_{32}, this node will output c_{22} and this node will output c_{23} and this output I will get during time t_7 and during time t_8, this node will output c_{33}. How the data will flow? If you analyze that, you will find that after time t_6 simultaneously I get c_{11} c_{12} c_{13} c_{21} and c_{31}. During time t_7, I get c_{22} c_{23} and c_{32} and during time t_8, I get c_{33}. So as against in this case where we have said that for this simple multiplication, matrix multiplication I need 27 operations. The same output is obtained in just 8 clock periods. So this gives another example how pipelining can reduce the computation time and this sort of pipeline has a specific name systolic array, you know that. These two types of pipeline architectures that we have discussed, one for floating point addition and the other for this matrix multiplication, you will find that these kinds of pipelines are dedicated pipelines. That means you can perform only one operation with this pipelines but it is ensured that operation is more efficient.

(Refer Slide Time: 00:20:20 min)



We can have other kinds of pipeline as well which are called generalized pipelines. What is the generalized pipelines? In both these kinds of pipelines, you have seen that the data flows only in one directions. That is from stage i, the data goes to the computation stage i plus 1. I don't have any provision of giving the data to a stage j where j is greater than i plus 1. From stage i, I cannot give the data to a stage j where j is greater than j plus 1 or j is less than i. That means I don't have any provision of feed forward data flow or I don't have provision of feedback data flow. So in case of generalized pipeline we can have the provision of both feed forward as well as feedback data flow.

(Refer Slide Time: 00:21:29 min)



So in this case of pipeline what we have done is or if you remember the earlier pipeline that we have done for floating point addition, we have always said that the data enters from one end of the pipeline, it comes out of the other end of the pipeline. From stage s_1 the data always goes to stage two. From stage two it always goes to stage three, from three it always goes to four. I don't have any facility of sending the data from stage one to stage three. I am always sending the data from stage one to stage two or I don't have any facility of giving the data from say stage three to stage one. That is I don't have any provision of feedback connection. Here in this case from this stage, the data is always moving in this direction. From here the data always goes here, the data always goes here that means it always goes in the forward direction following a linear path. I cannot bypass of any this. I don't have any provision of sending the data from this processing element to this processing element directly. I always have go to where this processing element. So this is a kind of pipeline which is also known as linear pipeline. The data always flows in the linear direction.

In a generalized pipeline what we will see is from any stage i, I can send the data to any other stage j where j can be greater than i plus 1. In case of linear pipeline, j is always equal to i plus 1. form i, I can send the data to j where j is always i plus 1 equal to i plus 1 in case of linear pipeline. In case of a generalize pipeline, j can be greater than i plus 1 that means I can bypass some of the stages in between and feed forward the data or j can also be less than i that means I can send the data in the backward direction in the pipeline. So such a generalize pipeline can have a structure like this. Suppose we have got three spaced pipeline s_1, s_2 and s_3. If it is a linear pipeline, always data will move from s_1 to s_2, s_2 to s_3 and finally you get the output.

Now if it is not a linear pipeline and we have also said that to take care of the speed mismatch, we can put latches in between stages. So let us also put the latches. We will have one latch here, one latch here, one latch here and finally one latch will come here. From the latch, the data always go to the corresponding stage. I will have an input, this stage will give an output, this will also give an output, this will also give an output but now I have to have the provision of selecting the source of data. So as I said that in this generalize pipeline, in case of linear pipeline the data will always move from s_1 to s_2. So this output of s_1 can be directly connected to the input of this latch. Similarly the data will flow from s_2 to s_3, so this output of s_2 will be directly connected to the input to this latch but now in case of generalized pipeline, I can also have a provision that from s_1 I should be able to send the data to s_3 bypassing s_2.

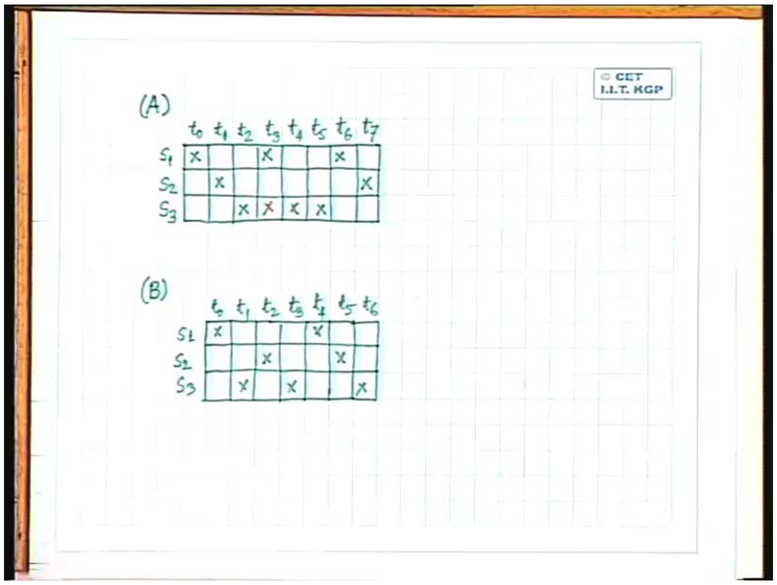
Similarly from output of s_3 I should be able to send the data to s_2, I should also be able to send the data to s_1 which will be the feedback connection. If I send the data from s_1 to s_3 that becomes a feed forward connection. from s_1 to s_2 or s_2 to s_3 that becomes a linear connection. So all these three types of connections should be possible in a generalized pipeline. so because now the source of data to every stage can be one of many, so I should put some sort of multiplexer using which I can select the source of data to the input of every latch. So what I can do is I can put a multiplexer here. One input of this multiplexer is the input data and output of the multiplexer is coming to the input of this latch. The other inputs to this multiplexer can be from output of s_3.

Similarly I will also put a multiplexer here that is input to s_2 can also come from the output of s_3. So like this and similarly s_3 can get input from s_2, it can get input from s_1. It can also get input from s_3 itself that is the self-loop. So in this case I can do a connection of this form s_1, the data can go to s_3, the data can come to s_3 from s_3 itself. I can have data path something like this and each of these are multiplexers. This is a multiplexer, this is also a multiplexer, this is also a multiplexer. So you find that by properly selecting the select inputs of the multiplexers, I can send the data from output of s_1 to the input of s_3. I can send the data from output of s_3 to the input of s_1, I can also send the data from output of s_3 to the input of s_2. I can also send the data from output of s_3 to s_3 itself.

So I have a reconfigurable type of pipeline. Depending upon what is the function requirement, I can choose these multiplexer select inputs so that the data will be routed through this pipeline as required. For functions I can also perform, this can be used for computing a single function. It may also be used for computing multiple function. That is more than one functions can be computed by the same pipeline. So if I want to compute more than one functions then suppose I can take an output from this stage and this gives me a function A. If I take output from this point, it gives me a function B. So there may be other connection as well. For example in this case we have not shown a feedback connection from s_2 to s_1. That is also possible. With the help of this, I can only feedback the data from output of s_3 to s_1, I can feedback of data from output of s_3 to s_2, I can also feedback from s_3 to s_3 itself. I can also have a feedback connection from s_2 to s_1 or s_2 to s_2 itself that is also possible in a generalized pipeline.

Now given such a generalized pipeline, now we have to decide that because now it is reconfigurable. I have to decide that how the data path should be selected, how the data should flow to compute a particular function which is not required in case of a linear pipeline because in case of linear pipeline, the data flow is always fixed. From stage i it will go to stage i plus 1 but in this case because there can be various ways in which the data can flow from one stage to another, I had to specify for computation of a particular function, how the data path should be selected. So for computing any function, I have to specify what is called a reservation table. That means what are the time steps, what are the time periods during which, which of these computation stages will be used for computation of a particular function.

(Refer Slide Time: 00:32:08 min)



(A)

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
S_1	X			X			X	
S_2		X						X
S_3			X	X	X	X		

(B)

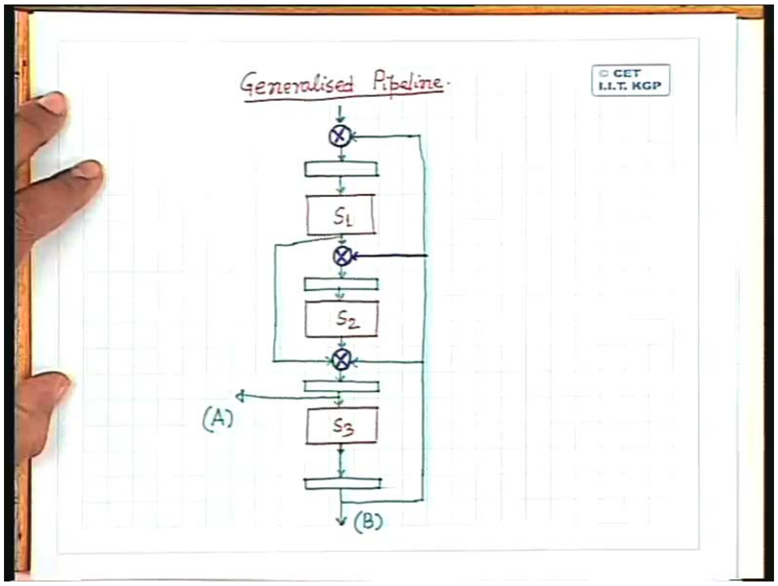
	t_0	t_1	t_2	t_3	t_4	t_5	t_6
S_1	X				X		
S_2			X			X	
S_3		X		X			X

So reservation table can be specified again in the form of a space time diagram. Say for example for function A, for the computation of function A my reservation table can look like this. There are three computation stages s_1 and s_2 and s_3 and these are the different time steps or time periods t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7. So when I specify the reservation table for function A, what I have to specify is during which of these time periods which of the computation stages will be used by this function A. So I can have a reservation table something like this. I can put a cross here indicating that during time interval t_0, it is the computation stage s_1 that will be used by this function. Similarly I can put a cross here, I can put a cross here. For s_2 it can be something like this, for s_3 it can be something like this.

Similarly to compute the function B, function B will also have a reservation table and that reservation table will obviously be different from this. If it is not different from this then obviously function A and function B are same. So for function B, the reservation table can be specified. Again I have the computation stages s_1 s_2 s_3, I have the time durations t_0 t_1 t_2 t_3 t_4 t_5 t_6, so something like this. So let us see how the data path will be set for computation of these two functions. [Conversation between professor and student (Refer Slide Time: 00:35:37)].

If the tables are same, output point is also same. Say depends upon during the last step, last time period which of the computation stages has been used? You will find that function for A, it takes 8 different time steps starting from t_0 to t_7. During t_7 the computation block that is used is s_2. That means for function A, s_2 is the output point. For function B at the last time step t_6, the computational block that is used is s_3 that means for function B, s_3 is the output point. For function A the output point is s_2, for function B the output point is s_3 so that is what has been shown in this pipeline diagram also.

(Refer Slide Time: 00:36:34 min)



I have taken output A, function A from the output of s_2 whereas function B has been taken from output of s_3. So as this reservation table says, it specifies that during which time step, during which time period which of the computational blocks are to be used for computation of a particular function. So here you find that for computation of say A during t_0, the computation block that is used is s_1 and this being the first one, the input obviously has to come from the external input. That means the data path that has to be active during t_0 is the direct data path from the external input to this latch and from the latch it directly goes to computation stage s_1.

Then you find that during t_1, the computation block that is used is s_2, during t_2 the computation block is that is used s_3. So these follows a linear path. At least for this first three timing durations, the data flow is linear. From s_1 the data will go to s_2, from s_2 the data will go to s_3 but after that during time interval t_3, the processing block that is being used is s_1 and prior to that, we had the output available from processing stage s_3. That means now I have to feedback the data from the output of s_3 to the input of s_1. So at this point the path that is to be active is this one and I have to set the multiplexer select input in such a way that now instead of taking the data from the external input, it has to take the data from this feedback path. From there data will come to s_1 and the corresponding computation will be performed.

After s_1 during time step t_4, you find the computational block that is to be used is block s_3. So now it does not follow the linear path anymore. In case of linear path here from t_0 to t_1 we had moved the data from s_1 to s_2 but now we have to move the data from s_1 to s_3. That means I have to make use of the feed forward path and this is the feed forward path. So you have to bypass s_2 and set the select input of this multiplexer in such a way that the data from this input will be selected and that will be fed to the computational block s_3 for this part of computation. This way the computation will continue and at the end of t_7, you get the output from the computation stage s_2.

Similarly we can also find out how the data flow will be performed for this function B. So it is this reservation table which specifies that how the data will flow through the pipeline and how many times it will be fed back or feed forward to compute a particular function. Obviously the pipeline control circuit which decides that how the multiplexer select input will be decided or how the data will be latched into different latches that will make use of this reservation table because it is only through the reservation table, the control circuit will know how the data has to move or how many times steps will be needed for computation of a particular function. This reservation table is very important for a generalized pipeline. They will have different delays. They will generate same delay. No. Blocks can have different delays but the final delay, I mean how much delay is allowed that is decided by the control circuit. That is the control circuit will which will decide that when to latch the data from one stage to the latch and that takes care of what is the maximum delay in different stages.

At a particular time step only one block is working for these functions but it is not necessary I can have a situation like this as well. This is also possible in a reservation table. That means for computation of function A during time step t_3, you are making use of both the computation stage s_1 and stage s_3. So if you do this, you will find that during time step t_2, s_3 was used. So after time step t_2 the output is available for s_3. During this time both s_1 and s_3 are being used. That means I have to set the multiplexer in such a way that is that output of s_3 goes to both s_1 as well as s_3 itself. To s_3 itself is going to s_1 also. Data output of s_1 (Refer Slide Time: 00:43:07) this is leaving an output okay then after that the stage which is used is s_3. You are talking about this multiplexer, may be some more function will be generated here. That is also possible instead of simple multiplexer I can have a functional block.

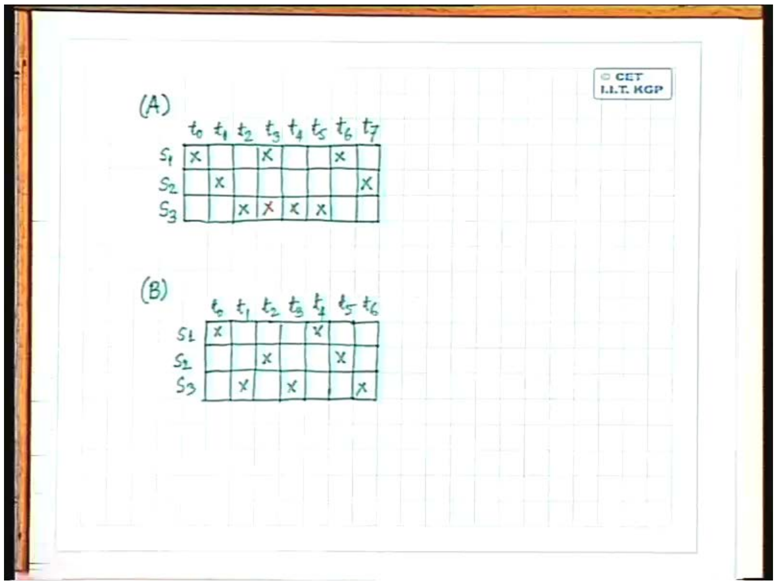
So multiplexer will be a part of that functionality also or may be before s_3 I have one computation stage which will combine these two outputs, output of s three and output of s_1. What I have shown is a very simplified situation but this can be complicated. So in case of generalized pipeline, it is possible that more than one computation stage will be used during the same instant of time. That can be done. Not only that, in a complicated situation it is also possible that the pipeline is being used for computation of more than one functions simultaneously, that is also possible. But how to do it that you have to find out. That has to be found out by analysis of the reservation table.

Now when I come to this generalized pipeline, here you find in case of linear pipeline I didn't have any problem. At every time step I can feed in a new data, a new task to the pipeline but when I come to this generalized pipeline that is not possible. By studying this reservation table, I have to find that what are the time steps, when I can feed in a new task to the pipeline. I cannot feed in a new task to the pipeline at every time step because in that case there can be data clash how? See for example here, for computation of a particular function f A, function A the pipeline makes use of this computation block s one during time instant t_8. It makes use of the same computation stage s_1 during time instant t three. It also makes use of the same computation stage s_1 during time instant t_6.

So if at t_3, during t_3 I want to feed in a new task, a new set of data but on which the same function A has to be computed. Then you will find that this t_3 becomes t_0 for that new task. So for the new task s_1 has to be used simultaneously, when s_1 is also being used by the previous task which is not possible. That means two task at time to use the same processing stage s_1 at the same instant of time. So obviously there will be clash. So that is what is called pipeline scheduling. I have to decide, when I can schedule a new task to the pipeline. I cannot schedule a new task to the generalized pipeline every time which I can do in case of a linear pipeline because in case of a linear pipeline, I know that the data enters one end of the pipeline, it goes out of the other end of the pipeline and in between it always has a linear flow. That means from s_1 it will always come to s_2.

For the same function, s_1 will not be used again. So at every time step, I can give a new task to a linear pipeline. I don't have to consider about this collision. I don't have to bother about this collision but in case of a generalized pipeline, I have to find out that what are the time durations or what are the time steps, when I can feed in a new task to the pipeline. That I have to do by analysis of this reservation table. So now let us see how we can analyze this reservation table.

(Refer Slide Time: 00:47:49 min)



(A)

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
S_1	x			x			x	
S_2		x						x
S_3			x	x	x	x		

(B)

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
S_1	x				x		
S_2			x			x	
S_3		x		x			x

So one thing I have said that here if I initiate a task at t_0, I cannot initiate next task at time instant t_3 because then there will be clash for this process stabilized one. Neither I can initiate a task at time instant t_6 because then also there will be a clash on s_1. Similarly here I cannot initiate at a task at t_1 sorry. If at some instant of time at t_1, this function A makes use of this processing stage s_2, I cannot initiate a task in such a way that the next task will make use of the same processing stage s_2 during time instant t_7 because then there will be a clash in this processing stage s_2. So I have to analysis this reservation table so that at no computation stage, there is a clash. So only when I know that by initiating a task, it does not lead to any clash I can initiate a new task at that time instant. That yet can be founded only by analysis of this reservation table. That we will do in the next part.