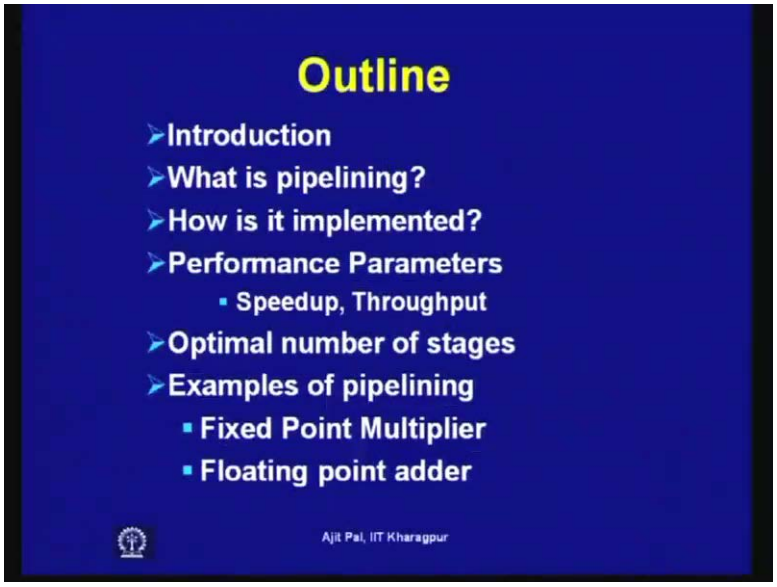
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 6
Pipelining – Introduction

Hello and welcome to today's lecture on Pipelining, pipelining is one of the most important and popular technique that is used to enhance the performance of a processor. And you will see that it is an implementation technique, which is done in the hardware and it exploits different types of a kind of parallelism, instruction level parallelism. So, before I go into the details of implementation of pipelining in modern processor, today I shall introduce to you, the basic concepts of pipelining. As you shall see, pipelining is a concept that is used not only in processors, but in our day to day common life in various situations and various applications.

(Refer Slide Time: 01:59)



So, it is necessary to understand the basic concepts of pipelining, and in this lecture I shall cover these topics. After a giving a brief introduction, I shall discuss about what is pipelining, I shall define pipelining, and then I shall discuss about it is implementation, how is it implemented. And whenever you implement pipelining, how the performance changes and usually the performance is represented by two parameters, speedup and

throughput. We shall see, how this speedup and throughput changes, as you implement pipelining.

And we shall see that, whenever you go for pipelining, it is implemented by defining a particular operation and it is implemented in a number of stages and we shall see that, the performance is dependent on the number of stages. So, you have to indentify an optimal number of stages, that gives you good throughput and speedup based on cost performance.

Then, I shall discuss about two important pipelining that is, your fixed point multiplier pipelining and floating point adder pipelining, which are used in implementing pipelining arithmetic unit that means, whenever you are performing multiplication of fixed point numbers. And also as you know, multiplication takes longer time, so pipelining is, whenever it is implemented, it will improve the performance. Similarly, floating point addition is another very important operation, where pipelining can be implemented to improve the speed of operation.

(Refer Slide Time: 03:57)

Oh my god, a house is on fire, so as it appears, the house is in a village, where the fire brigade is not present and fortunately as we can see, there is a pond nearby. Let us see, how this fire is doused by this person, so obviously what he will do, he will take water from the pond and again come back to the pond to take water and then again put it on the

fire and so on. So, he will run between the house on fire and the pond to douse the fire off, so put the fire off.

This is how he can try to do it, but unfortunately this thing, they have that putting the fire off will take very long time. The poor fellow may lose his house, the house may be completely brunt off by the time he is able to put the fire off. So, let us see, how the fire can be put off in an alternative manner, which was taught in our school. In our school it was taught that, whenever there is a house on fire, you get hold of as many buckets as possible, get as many people as possible, let them stand one after the other between the pond and the house on fire. And then let the bucket full of water pass from one hand to another hand from the pond to the fire.

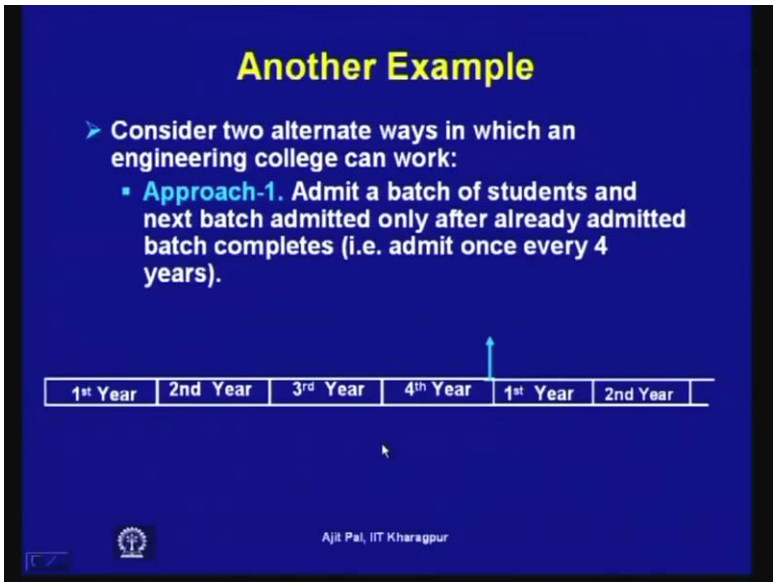
(Refer Slide Time: 06:00)

And the way as I shall show in this particular diagram, so you can see here, the bucket full of water is moving from the pond, it has been passed on to the next person. And in the mean time, the first person, the person who is near the pond again take another bucket full of water. And in this way as you can see, bucket is moving from the pond to the fire and you can see, several bucket 1 2 3 4 in this diagram and maybe 5, so 5 buckets of water is moving from the pond to the fire in parallel simultaneously.

And later on, when I understood I mean, when pipelining was taught I realized that, this is nothing but pipelining. So, what was taught in our village school for putting the fire

off, is a pipelining technique. So, this is one common example, one example of I mean, pipelining in our day to day life.

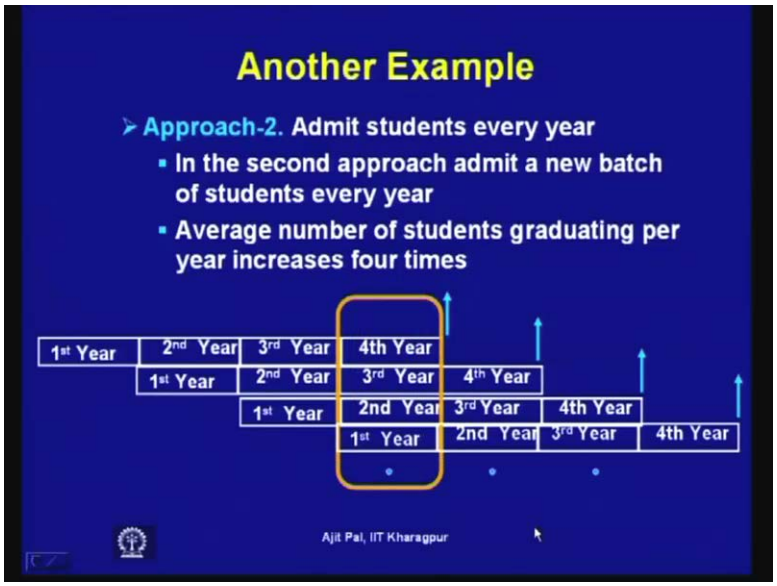
(Refer Slide Time: 07:15)



Now, let us see another example, so here you see, the two alternative ways in which an engineering college can run. Let us consider the first approach, in which admission take place only when the batch of student passes out from the college. So, you can see, the student is taken, he joins the 1st year course then he goes to moves to the 2nd year course then he goes to 3 rd year course then he goes to a 4th year course and at the end, of the first 4th year as he passes out, then only another batch of student is taken.

So, in this particular case as you can see, admission of student is taking place once in every four years. So, this is obviously not a very good way of doing it and in this particular case as you can see, the throughput that is, the number of student passing out per year is only 1 by 4. Because, only one student I mean, in every four year, one batch is passing out, so throughput is 1 by 4, obviously this is not practiced in our colleges.

(Refer Slide Time: 08:29)

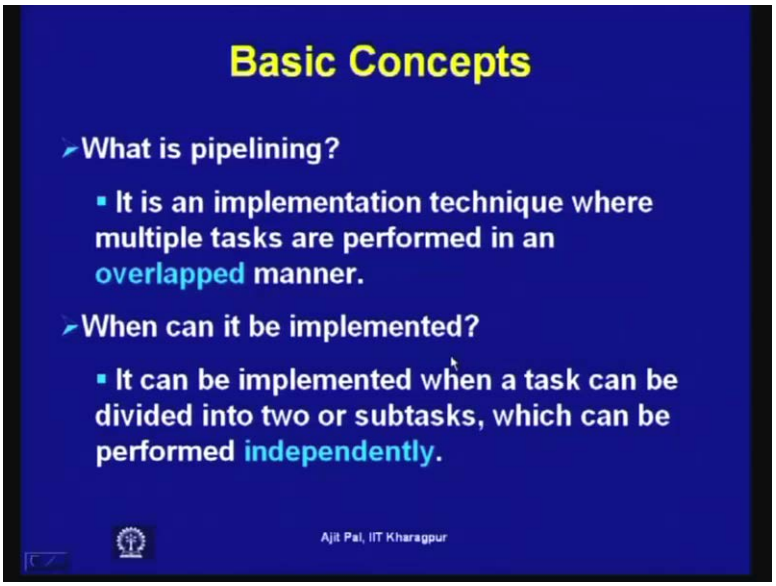


So, in our colleges what is being done, student is admitted every year, so you can see, as the first batch of students moves to the second year, another batch of student is admitted. And as the second batch of student moves to third year and the second batch of students moves to the second year, another third batch is taken. And the first batch moves to the 3rd year to 4th year, second batch moves from 2nd year to 3rd year and third batch moves to 1st year to 2nd year and a fourth batch is admitted.

So, you can see, at the end of 4th year, initially there is some delay, at the end of 4th year only first batch is coming out. But, subsequently as you can see, in the 5th year, another batch is coming out that means, the second batch is coming out in the 5th year, third batch is coming out in the 6th year and so on. So, if you whenever a college is running for a large number of years, you can ignore the first initial few years.

And then you will see that, the college is able to produce students one batch per year, so throughput is increasing from one fourth to 1. So, this is also a technique for pipelining, so we have seen two I mean, applications of pipelining in our common life, day to day life.

(Refer Slide Time: 09:54)



Now, let us see, what is the basic concept behind pipelining, so whenever you are asked define pipelining, you will say it is an implementation technique, where multiple tasks are performed in an overlapped manner. So, what has been done essentially, multiple tasks have been carried out in an overlap manner. So, if we go back to our previous diagram we see that, here the first batch who are in the 4th year is being taught simultaneously with the second batch of students, who are in the 3rd year.

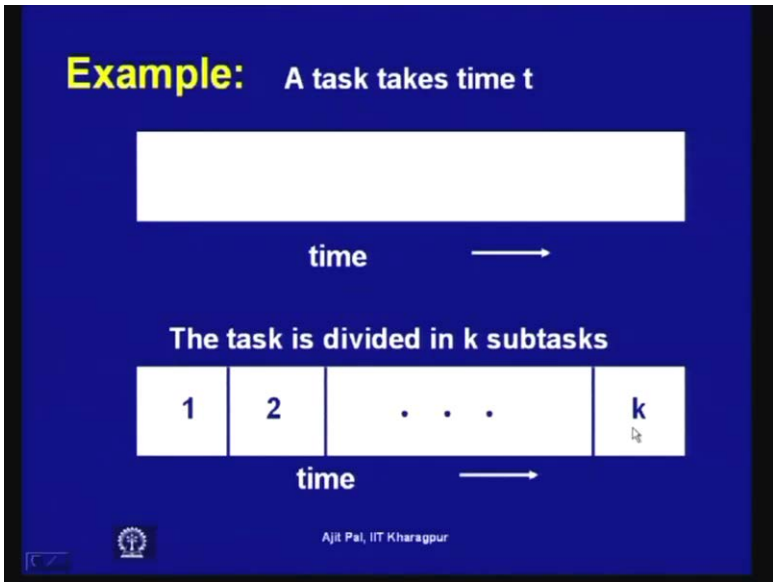
Similarly, a third batch of students were in the second year and a fourth batch, who are in the first year. So, all these batches of students of I mean, are taught simultaneously, although they are in different years and this is essentially the basic idea of pipelining, so in an overlap manner, multiple task are being performed. So, in this particular case, task is teaching students in a college, so it can be any kind of task.

Now, let us see, how can it be implemented, this question arises because as you have seen, throughput increase whenever pipelining is implemented, question actually arises, can it be implemented in all possible cases. Let us see when it can be done, so the answer is, it can be implemented when a task can be divided into two or more subtasks, which can be performed independently, so this is the key idea of for implementing pipelining.

First thing, first requirement is that, you should be able to divide a task into more than one subtasks 2 3 4, whatever it may be. Then it is also necessary that, these subtasks you should be able to perform independently. So, you have seen in case of college, the

teaching of students of different batches, who are in different years of study, can be carried out, provided we have enough infrastructure, building, classrooms and teachers then it can be done independently. And that is how pipelining is implemented in case of college, so that is the requirement for implementing pipelining.

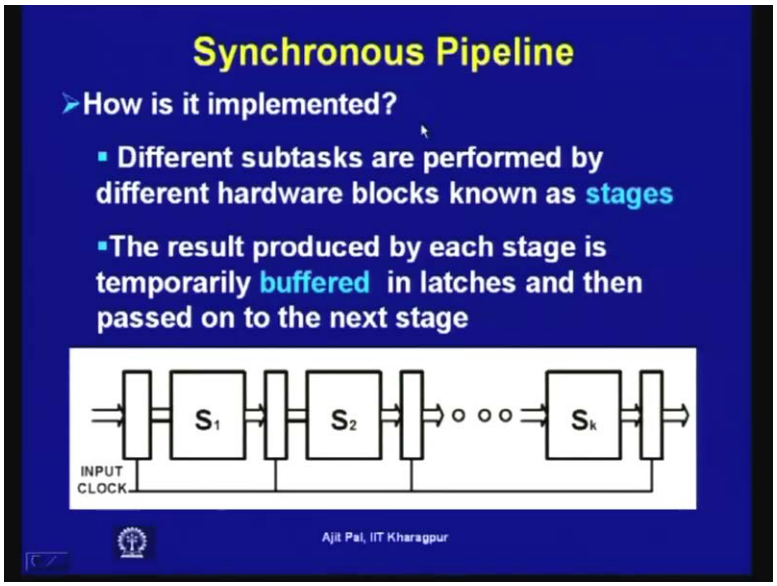
(Refer Slide Time: 12:31)



Now, let us see a task, which is taking time t, so this is a task, single task and it is taking time taking time t. Now, let us see, whenever it is divided into k subtasks, so here you have got a task, let us assume it takes time t. Now, whenever you divide it into k subtasks, so you have divided into k subtasks, obviously each of them now we will take time, which is equal to t by k, t by k is the time required to perform each subtask

Now, if you think in terms of implementing it, here the time required is t for a task and here time required is t by k to perform a subtask. We shall see, whenever we go for implementing this in the context of processors, they are the, how the clock frequency and other things are affected by this.

(Refer Slide Time: 13:55)



Now, the pipelining can be implemented in two ways, first one is known as synchronous pipeline, which is the most popular one and here different subtasks are performed by different hardware blocks known as stages. So, you have seen, we can divide a task into a number of subtask, so this is performed by a particular stage. So, t will require k stages and each will perform a particular subtask and obviously, these stages will perform different operations, because you have divided task into k subtasks.

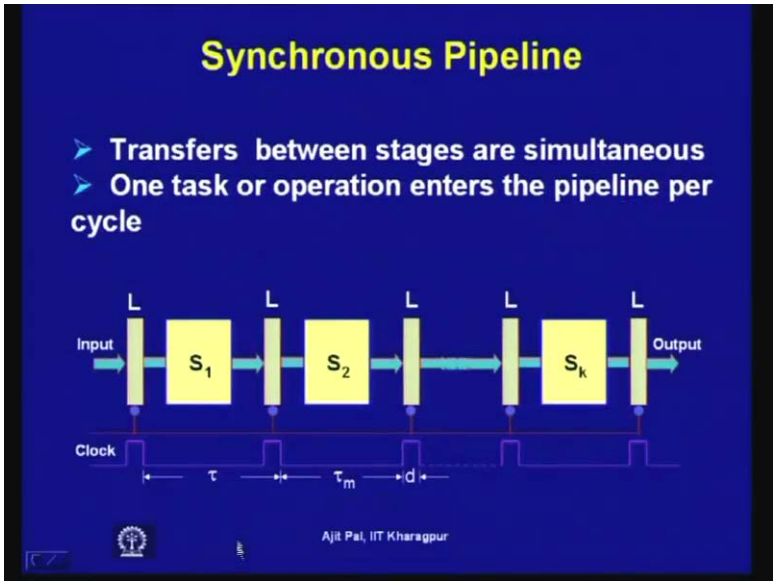
And each of this subtask will be different, they are not same and obviously, these stages will perform different operations. Now, different subtasks are performed by different hardware blocks known as stages and the result produced by each stage is temporarily buffered in latches and then passed on to the next stage. So, what is important here, not only you have to divide a task into k subtask and each of the I mean, subtask is performed by a particular stage.

But, it is also necessary that, you have to insert some kind of buffer in between each stage. Why is this necessary, this is necessary because whenever this stage will be performing some operation, it will get it is input maybe from the input. And then when the result is produced, that has to be temporarily stored in this particular latch or buffer. And subsequently, if an another task is applied to it, this latch will provide input to the second stage.

Similarly, the output produced by the second stage, will be buffered in this latch and which will provide input to the third stage. So, in this way, the different stages will get their inputs from the latches, only the first stage will get inputs from the primary input that is applied. So, and as you can see, we will also require a clock, this clock will latch the input, that input which is available into this latch. Similarly, the output produced by stage S 1, will be also simultaneously I mean, latched into this particular buffer by this clock.

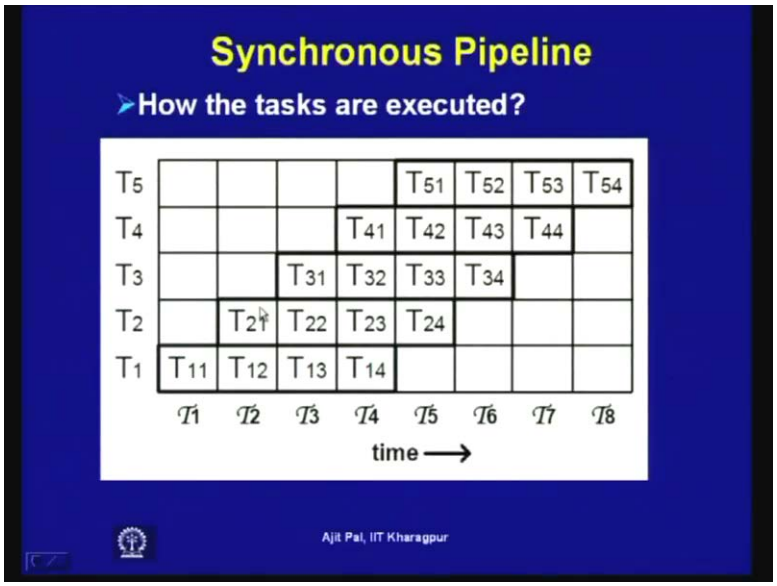
So, in this way, output of different stages are latched in buffers simultaneously with the help of a clock. So, as the inputs will be coming, the outputs will be generated and that will be latched and we shall see, how the input will flow from input to output, we shall explain it with the help of another diagram.

(Refer Slide Time: 17:13)



Now, transfer between stages are simultaneous, as I have mentioned and one task or operation enters the pipeline per cycle, this is a very important thing. See here, clock per cycle, a new input is coming and which is getting admitted into the pipeline. And you can see, the clock is applied to different latches, different I mean, latches in between a pair of stages.

(Refer Slide Time: 17:47)



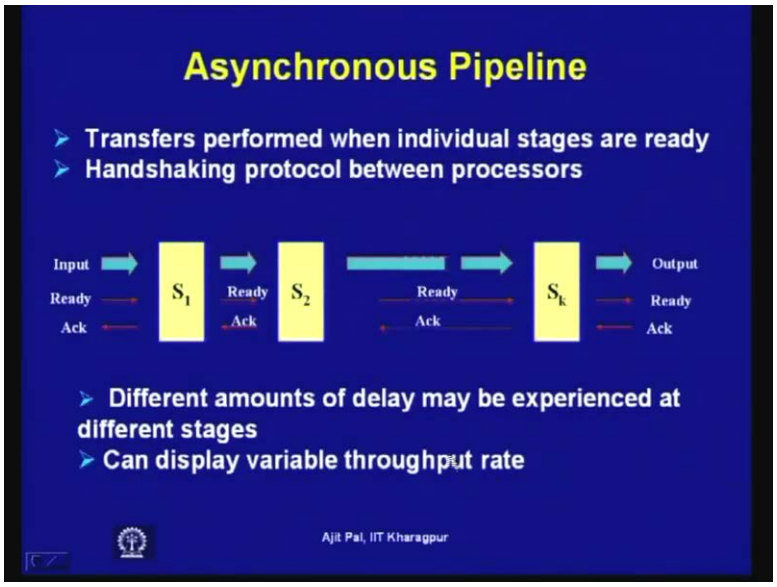
So, here let us see the, how the execution take place, so you can see here, in the first clock cycle, the task T 1 is entering the pipeline and entering the first stage of the pipeline. Then as we move to the second clock cycle then the another task is entering the first stage, but in the mean time, that output of the first stage has been latched in a buffer and which is being applied to the second stage. So, second stage and first stage, both are performing computation, performing some operation, but they are performing operation on the data of different tasks.

So, this stage 1 is performing operation on task inputs coming from task 2 and stage 2 is performing operation of that is, generated by the first stage, intermediate result produced by the first stage that is, buffered in between and latched and essentially, it is the data coming out from the task T 1. So, in this way you can see, as we move to the fourth clock cycle, all the stages are busy. The stage 4 is performing processing on data produced corresponding to task 1 then stage 2 I mean, stage 3 is producing it is performing operation on data that is coming from task 2.

Then, the third I mean, third task is being performed by stage 3 and stage 4 is performing the operation I mean, performing operation on task 4. So, you can see, this is how, in a overlap manner processing is taking place and now, at the end of fourth clock cycle, here it has been assumed that, it requires 4 stages. So, you can see, 4 clock cycles are required

before the first result comes out then in every clock cycle, a new output is being generated.

(Refer Slide Time: 20:04)



Question naturally arises, we have defined synchronous pipeline, so there must be an alternative technique, that is known as asynchronous pipeline. So, in an asynchronous pipeline, transfers performed when individual stages are ready. So, here you see, we are not providing a latch in between, but there is a kind of handshaking signal present between two stages. You can see, there is a ready input and acknowledgement input, so ready signal is coming as input to stage S 1.

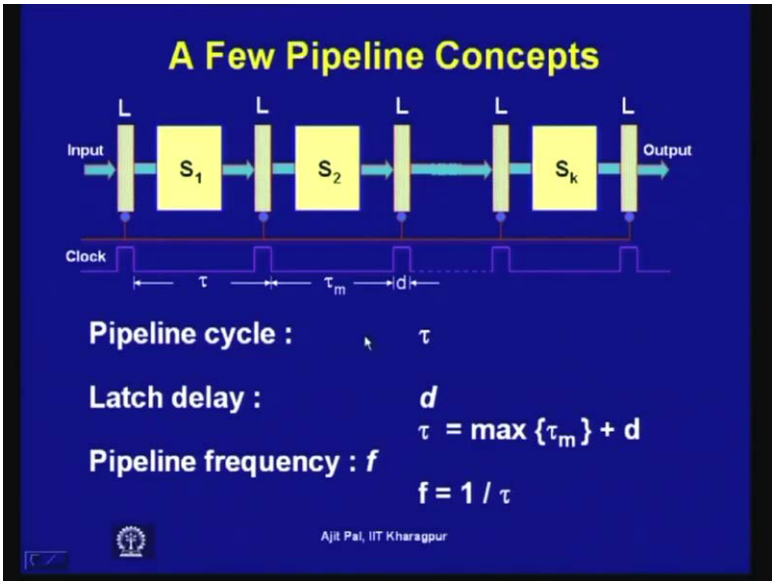
And whenever it is ready to accept some new data, it will generate an acknowledgement signal and then new input will be provided to stage S 1. Similarly, stage 2 will receive that input from stage S 1 and only when acknowledgement will be sent by stage 2 to stage 1. So, in this way you can see, the data will be passed on from one stage to another stage with the help of handshaking signal in an asynchronous manner, there is no clock.

Obviously in this particular case, the delay I mean, the time required to perform operations of different stages can be different. In other words, the different amount of delay may be experienced at different stages. So, in this particular case, say stage S 1 may take say 5 nanoseconds, stage 2 may take 10 nanosecond and so on, variable time, so and as a consequence what can happen, it can display variable throughput rate. On the

other hand, in our previous case we have seen, in case of synchronous pipeline, that is not so.

A common clock is used to move data from one stage to another stage and as a consequence, the throughput is fixed determined by the clock rate, the rate at which output is produced. And also the delay that is been, we assume that, each stage take same time that means, the delay taken by different stages is same, there is no difference.

(Refer Slide Time: 22:30)



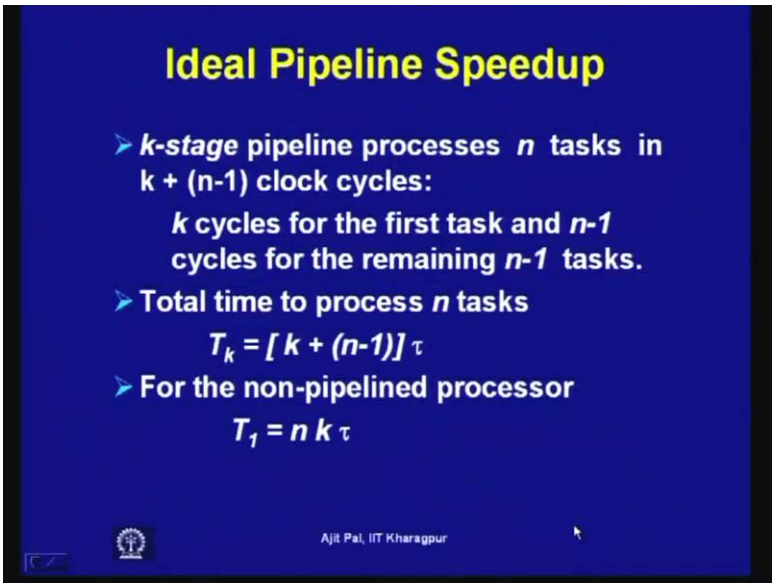
However, it I mean, this cannot be achieved in practical situations, so let us see, whenever the different stages, say here you have got stage S 1 and here is another stage S 2. And there is a latch in between and another latch in between stage S 2 and S 3 and in this way, you have got a k stages. Now, you will be applying a common clock to all the latches and at the input also, this is the clock. How do you decide the clock or identify the clock frequency, actually it is dependent on the cycle time or time required to perform the operations of different stages.

And in case the time required is different, let us assume this stage S 1 takes time T 1, stage 2 takes time T 2, stage 3 take time T 3, stage k takes time T k. So, in such a case, how do you find out the cycle time that is equal to tau, how do you find out that. What is being done, the clock cycle time has to be the worst case cycle time of a particular stage. So, what you will do, you will take maximum of the tau m, let us assume the stage m has the maximum delay.

So, one of these stages will have maximum delay and that time has to be taken and also you have to take small delay, that is taken by these latches, maybe d, that is the delay of the latch and this is how, the cycle time of a pipeline implementation is decided. Now, cycle time is tau and obviously, the clock frequency can be derived from this clock frequency, is equal to 1 by tau.

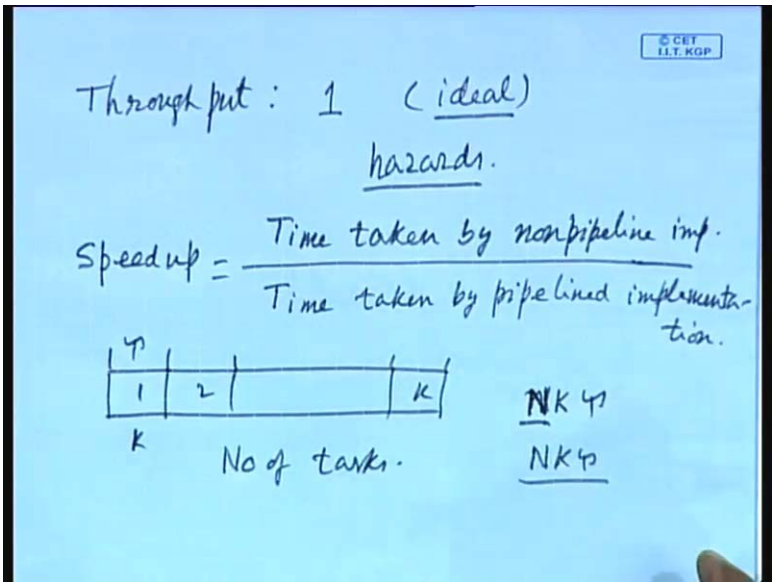
So, this will be the clock frequency that has to be apply here, so this is the time period then this is equal to tau and clock frequency is equal to 1 by tau. And based on this, the clock frequency will be decided, so worst case delay of a stage will decide the clock frequency.

(Refer Slide Time: 25:26)



Now, let us see, how the speedup and throughput changes.

(Refer Slide Time: 25:39)

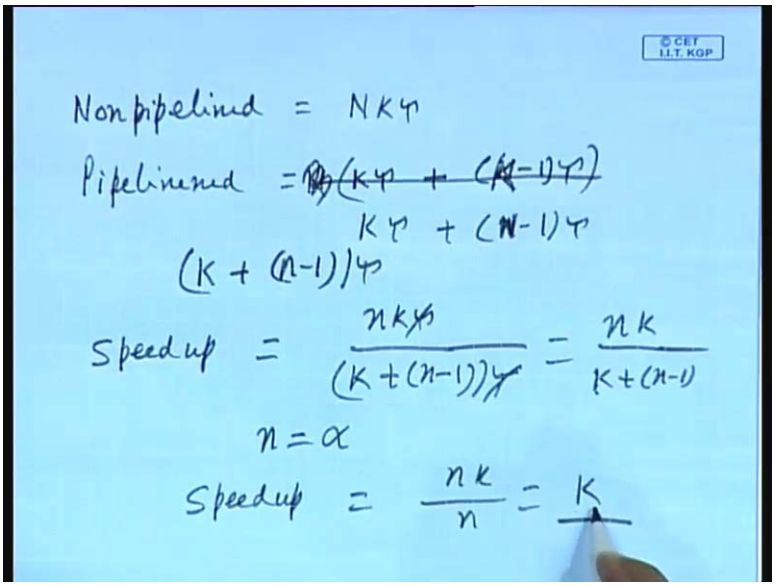


As you have seen, throughput is the outputs produced per clock cycle and that throughput will be equal to 1, in case of ideal situation that means, when the pipeline is producing one output per clock cycle. And later on we shall see that, there are I mean, this will not be I mean, there will be situations, where the output cannot be produced in each cycle and because of problems know as hazards. So, later on, we shall discuss about pipeline hazards, for the time being let us assume, we are considering, we are dealing with ideal pipelining.

Now, how do you compute the speedup, speedup can be found out from the time taken, it will be the ratio of time taken by non pipeline stage, non pipeline implementation and the speedup, the time taken by pipeline implementation. How do you find it out, if you have seen that, you are dividing a task into k sub tasks, 1 to k. Now, let us consider that, each of these times are identical same and then the time required, that will be equal to N into k into tau, assuming that this is equal to tau.

Because, to perform a task of, where N is the number of tasks, here we are not considering one task, time taken by pipeline implementation to perform n tasks. So, one task will take k tau time and in a non pipeline implementation, an N tasks will take N k tau time, unit maybe nanosecond, millisecond, whatever it maybe.

(Refer Slide Time: 28:40)



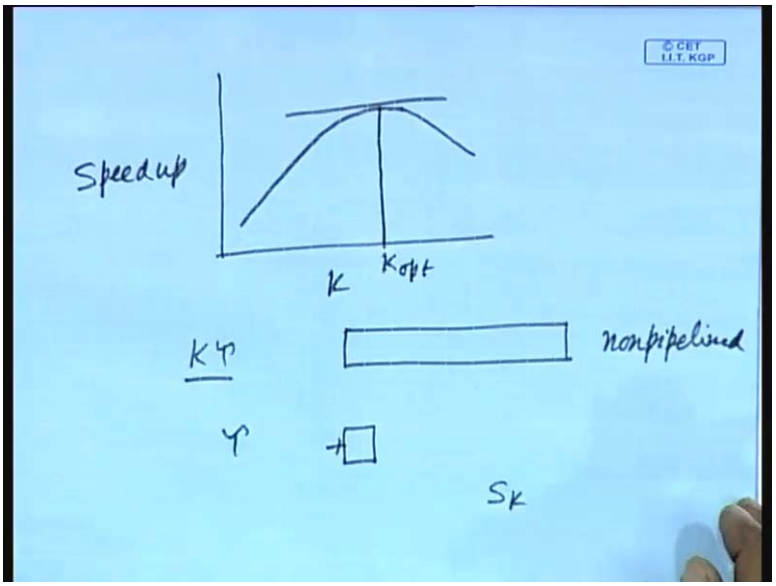
Now, what is the time taken, so non pipeline is, non pipeline implementation is taking time N k into tau. How much time will be required by pipeline implementation, first of all, in case of pipeline implementation we find that, the first task will require k into tau time and the remaining tasks N minus k 1 tasks, will produce in tau time. So, you can see that, so here (Refer Time: 29:33) it will take k minus 1 time. And if you have got N tasks then it will be performed in this way.

So, it will be taking k plus n minus 1 into tau, not this here it will be, I made a mistake, here it will be k into tau plus N minus 1, N is capital or small, whatever it maybe into tau, so k plus n minus 1 into tau. So, what is the speedup, speedup is equal to n k tau by k plus n minus 1 into tau and this tau tau will cancel out. So, you will get here, n k by k plus n minus 1, so this is a very simplified expression. Now, let us assume, n is infinity that means, when you are performing a large number of tasks then what will happen, n will be equal to infinity.

o, in such a case, your speedup will be equal to n k and here, you can ignore k minus 1 with respect to n, so it will be equal to n, so this will be equal to k. So, we find that, the number of stages that will be required is equal to k I mean, is equal to k that is, speedup is equal to k and which is equal to number of stages. So, in other words, we are getting a speedup in ideal situation, that is equal to number of stages.

So obviously, you will be tempted to use as many stages as possible, but unfortunately later on we shall see, it is not true. Because, as the number of stages increases, overhead keeps on increasing, you have to put on buffers and a point is reached when you will not get any more speedup.

(Refer Slide Time: 32:06)



So, the speedup will initially keep on increasing, but the curve will be somewhat like this, say if we plot speedup and here the value of k. We will find that, initially it keeps on increasing but then it will decrease. So, the point will reach when you will get an optimum speedup, so that speedup will have always some optimum value (Refer Time: 32:35).

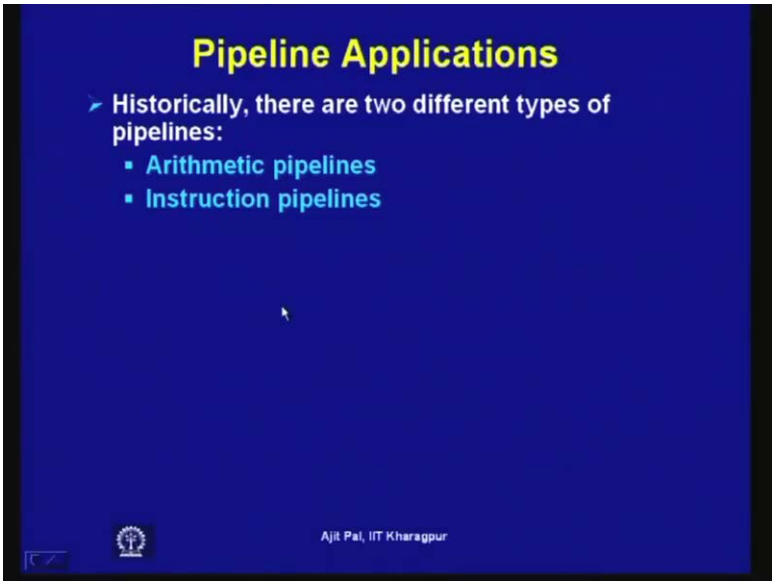
Another point that you have to remember that is, the memory bandwidth, we have seen that, in case of non pipeline processor, the rate at which data or instruction, whatever it maybe is transferred, the rate is k tau that is, the time required to perform a single operation. So, if it is a processor performing I mean, executing instruction, each at the interval of k tau, one instruction is fetched from the memory.

Because, CPU will be fetching the information from the memory, instructions from the memory and for a non pipeline processor, the rate will k tau, where k is the number of stages and tau is the time required per stage, so this is the case in case of non pipeline. On the other hand, in case of pipeline stages, you have to feed data at the rate of tau that means, the input has to come at the rate of, because the new input has to be provided to

the processor at the rate of tau in each cycle of the clock cycle of the pipeline processor, your new input will come.

So, the bandwidth of the memory has also to be increased by factor of S k, where S k is the speed of factor. That means, what I am trying to tell, not only the clock frequency of the processor will be k times that of a non pipeline processor, the bandwidth of the memory is also I mean, that is required will be the S k times that is, the speed of times that of the non pipeline processor. In other words, you will require a faster memory, whenever you use it in the context of a pipeline processor.

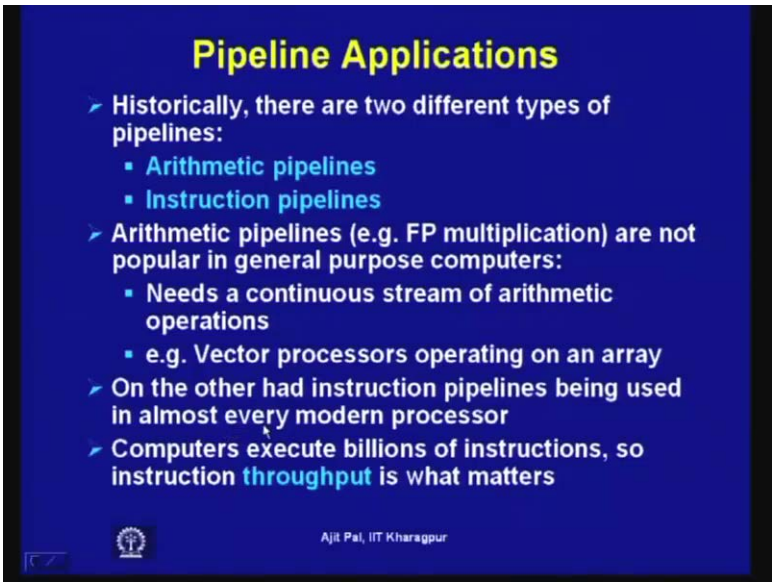
(Refer Slide Time: 34:48)



Now, what are the different types of pipelines that is possible, historically there are two different types of pipelines, one is known as arithmetic pipeline, another is instruction pipeline. So, in arithmetic pipeline, you perform different types of arithmetic operations like addition, subtraction, multiplication and division. It may be fixed point addition, subtraction, multiplication, division or floating point addition, subtraction, multiplication, division, so these operations can be performed by the arithmetic pipelines.

On the other hand, in instruction pipeline, the instructions will be fetched from the memory and the instructions will be executed in different stages of pipeline. So, the ALU operations can be performed by arithmetic pipeline and instruction processing can be performed I mean, fetching, execution can be performed by instruction pipeline.

(Refer Slide Time: 35:51)



Now, arithmetic pipelines like floating point multiplications are popular in general purpose computers and question actually arises, when a pipelining can be used. We have seen that, pipelining can be used whenever you are getting continuous input and producing continuous output. For a single task, it does not give you any benefit, the reason for that is, for a single task, time required to perform computation execution is more than that of a non pipeline implementation. That means, only when large number of tasks are to be performed then pipeline is suitable.

And in case of arithmetic pipelines, we have seen, this is not very common, we do not keep on performing addition or subtraction or multiplication continuously. So, whenever an arithmetic addition operation is performed, that is performed and maybe another addition will be performed after many clock cycles, so it is not very common. However, there are situations, where you have to perform large number of operations and that situation is for example, vector processors.

So, where you have to, an array has to processed and that number of array elements can be large, which is usually performed by vector processor, that type of thing can be done, can be performed by arithmetic pipelines. But, so far, as instruction pipeline is concerned, the instruction pipelines are being used in almost every modern processor. The reason for that is, we have seen that, whenever a processor of a computer is turned

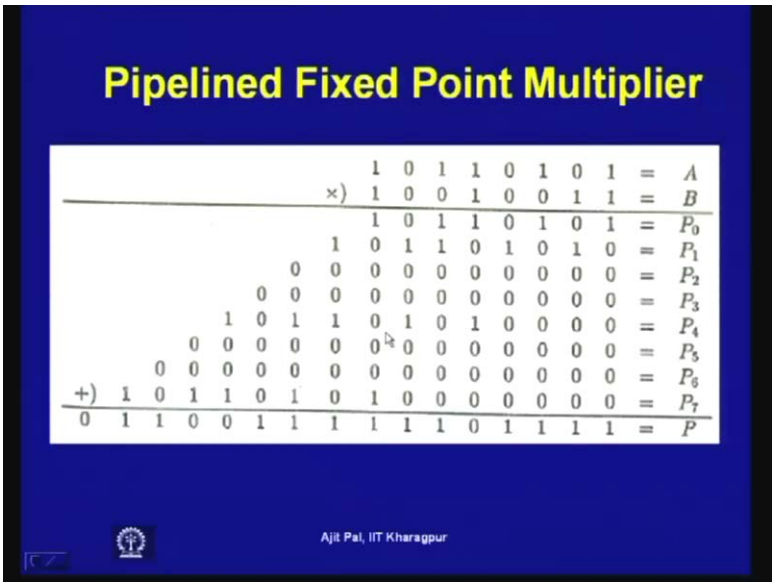
on, all it does is fetching an instruction from the memory and executing it, fetch execute, fetch execute and it does it continuously one after the other.

And you have got a constant stream of instructions, that is stored in your computer memory and those are being fetched one after the other, as long as the power is on. So, instruction pipeline is a very good case I mean, implementing pipeline for instruction is a good case and that is the reason why, instruction pipelining is performed in all modern processors. But, before we go for instruction pipelining, for the sake of completeness, we should discuss about arithmetic pipelines with two examples.

(Refer Slide Time: 38:43)

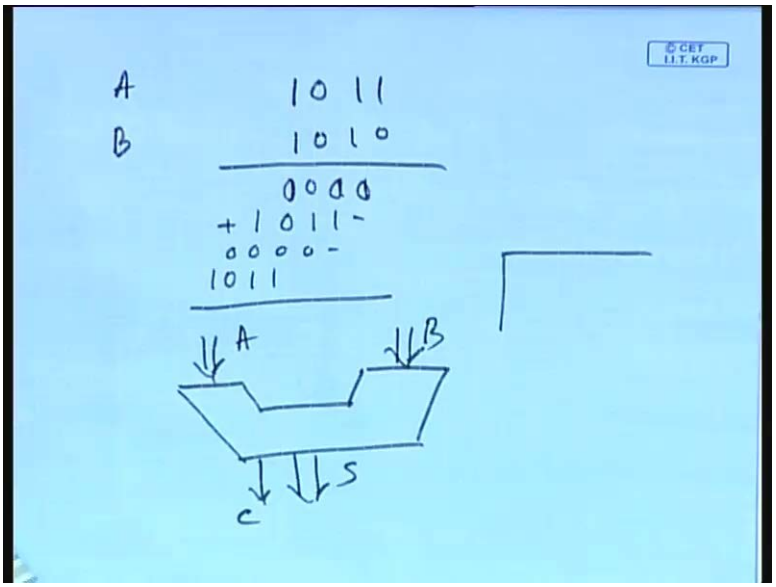
First one is pipelined fixed point multiplier and then later on we shall discuss about pipelined floating point adder.

(Refer Slide Time: 39:03)



So, first let us focus on, how pipelining can be implemented for a fixed point multiplier.

(Refer Slide Time: 39:10)



So, whenever you are performing multiplication of two numbers, say A and B, say is equal to let us assume 1011 and the number another number is 1010. So, whenever you have to multiplier, what you do, whenever you do by hand computation, you first multiply 0 with this, so you get 1011 then multiply 1 with, (Refer Time: 39:33) you multiply with 0, you will get 0000. Whenever you multiply with 1, you get 1011 and you

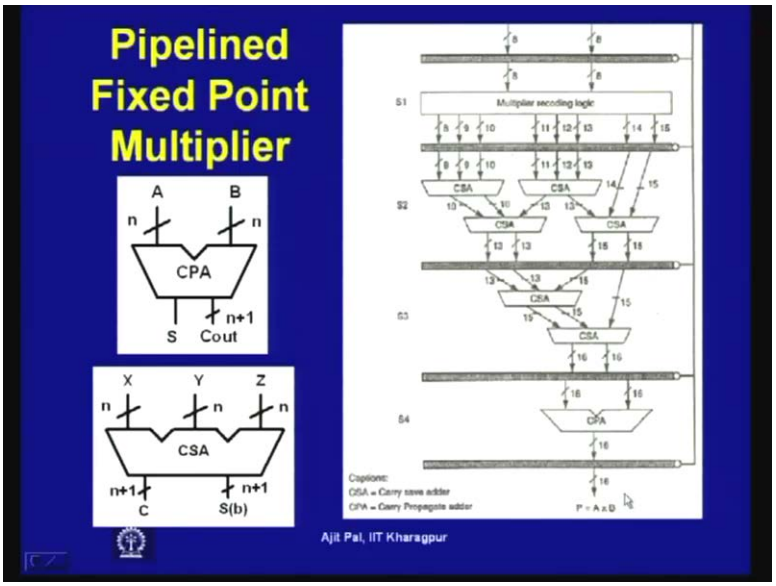
keep on shifting it and you essentially, we do shift and add, shift and you have to add these two numbers.

You have perform addition of this and then whenever you multiply with this, again you get 0000 and then another is 1011. So now, these are added, so this addition whenever you perform by hand, we may do it simultaneously that means, we add all these bits together and produce the sum, as it is shown in this diagram (Refer Time: 40:17) here, A and B, two 8 bit numbers, P 0 is a partial product. By multiplying this bit one with this number, all the bits, we get this and in this way, we have got partial product P 0, P 1, P 2, P 3, P 4, P 5, P 6 and P 7 corresponding to multiplication of these 8 bits with these bits.

So, these are the partial products, which are to be added and you can see 0 has been inserted on the right side. And whenever we do the addition of the entire column, but whenever you do it with the help of practical I mean processors, your adder will take two numbers, usual symbol of adder is this, adder will take two numbers and produces sum and a carry. So, this A, B, sum and carry, so you cannot really perform all these additions simultaneously, which is shown in this diagram.

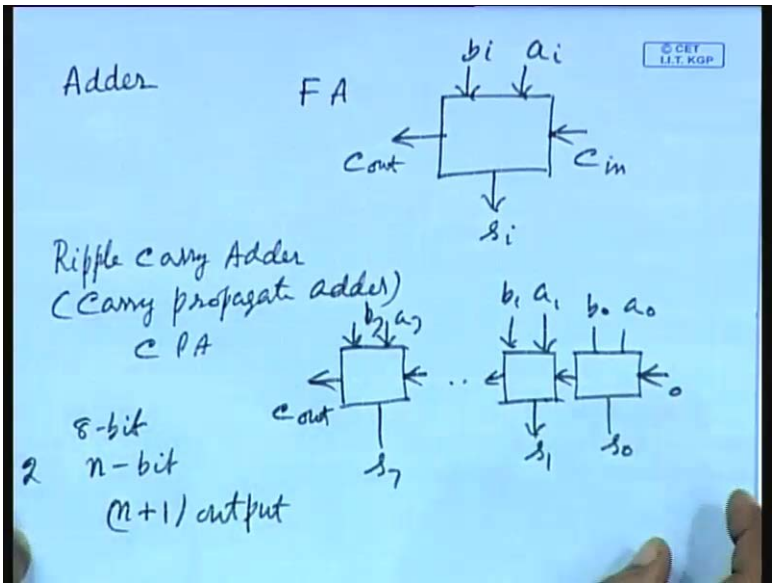
However, we can try to do it in a pipelined manner to I mean, instead of doing it serially that means, adding these two P 0 and P 1 then P 0 and P 1 then the result of that is added with P 2. And then result of that is added with P 3 and so on, that is essentially doing it serially, that will take long time. It will take definitely 8 clock cycles, if we perform one addition per clock cycle.

(Refer Slide Time: 42:11)



Let us see how this time can be reduced, whenever we go for pipeline implementation.

(Refer Slide Time: 42:19)

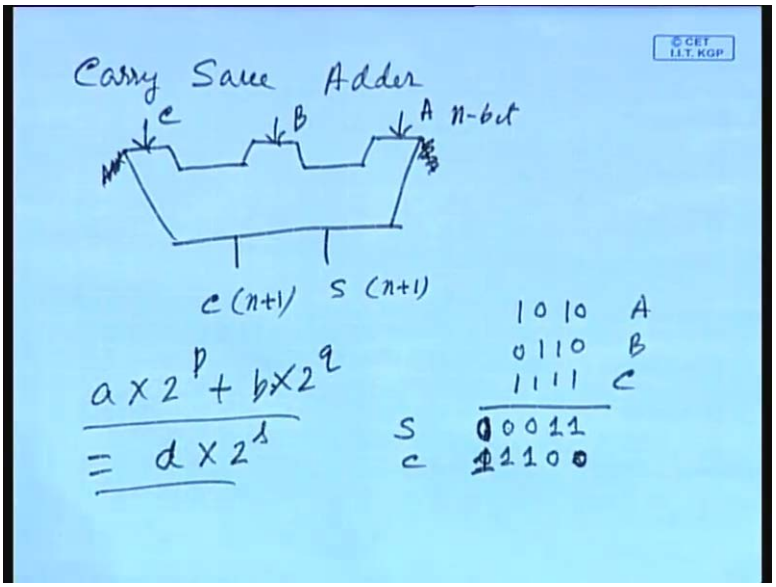


So, whenever we go for pipeline implantation, we can have two types of adders, you are familiar with full adder. A full adder takes 2 bit inputs, maybe a i b i and a carry C in and it produces a carry outs C out and sum bit s i. So, this is the full adder implementation that means, it is adding 3 bits, one is carry in coming from the previous stage and two bits and producing sum and carry. Now, you can realize two types of adder, one is known as ripple carry adder, which is also known as carry propagate adder that is, CPA.

How is it implemented, here you will have several full adders, if it is a 8 bit, you will have 8 bit full adder. So, here you will put a 0 b 0 then a 1 b 1, here you will put a 7 b 7 and this maybe 0 and the carry coming out from this will come here and s 0 you will get here, s 1 you will get here and s 7 you will get. So, you can see that, this is the final carry that will be generated, C out and the other carries are passing through from one stage to another, passing from one stage to another.

So, it ripples through and you get, you can see if this a two 8 bit numbers or say two n bit numbers, produces n plus 1 bit output. So, this is the case for carry propagate adder, so two n bit number is input and output n plus 1 bit output.

(Refer Slide Time: 44:43)



Now, there is another type of adder, which you will require to implement the pipeline implementation, that is known as carry save adder. Carry save adder has got 3 inputs, so here you have got 3 inputs, it can take 3 inputs simultaneously and here 1 bit, here another bit, here another bit. So, and what it does, it produces two n bit output, so say it will take one input A, another input B, another input C, each of them maybe n bit and it produces 2 outputs, one is sum, which is n plus 1 bit and another is carry, which is also n plus 1 bit.

How it is being done, let me illustrate this with the help of three 4 bit numbers, say 1010 another is 0110 and third number is 1111. So, whenever you perform these three numbers, so this is A, this is B and this is C. So, sum will be generated, which is a sum of

these numbers and as you know, sum is exclusive of all these three, so sum will be 1, sum will be in this case 1, sum will be 0 here, sum will be 0 here. And corresponding carry will be, if there is carry from this stage in this case and in this case also, there is a sum this will produce 1 here, so this is the sum.

And so far as the carry is concerned, so you can see, you have got 4 bit addition you are taking, so you have got 5 bit. Similarly, for carry, this bit will not be there, because carry be from the next stage. So, from this stage, there is no carry so it will be 0, from this stage there is a carry, so there will be 1. From this stage there is a carry, there will be 1 and from this stage, there will be a carry, so there will be 0. So, each of this stage will produce, so here also you require 5 bits, so you can see that, sum and carry both are requiring 5bits, so n plus 1, so this will be 0.

And in this case, this sum will be 0, there will be sum in this case 0 and in this case 0, so this will be a carry here, carry here, not sum, sum is 0, so carry is here. So, this is the most significant bit is 0 in this case and here, less significant bit is 0 for carry. So, this is how, two n bit numbers are produced by the carry save adder (Refer Time: 48:05). So, we can combine carry sum adder and carry propagate adder to realize a pipeline fixed point multiplier. As you can see, the first stage is producing the multiplier recording logic.

So, what it does, it produces (Refer Time: 48:22) these bits, this partial product bits, essentially by (Refer Time: 48:28) large number of AND gates. So, by large number of AND gates, you can produce these outputs and you can see, two 8 bit numbers, here you have got the corresponding partial products. So, last one was 15 bit and the first one was eight bit, which is shown here. So, you can see, here it is 8 bit then 9 bit, 10 bit upto 15 bit.

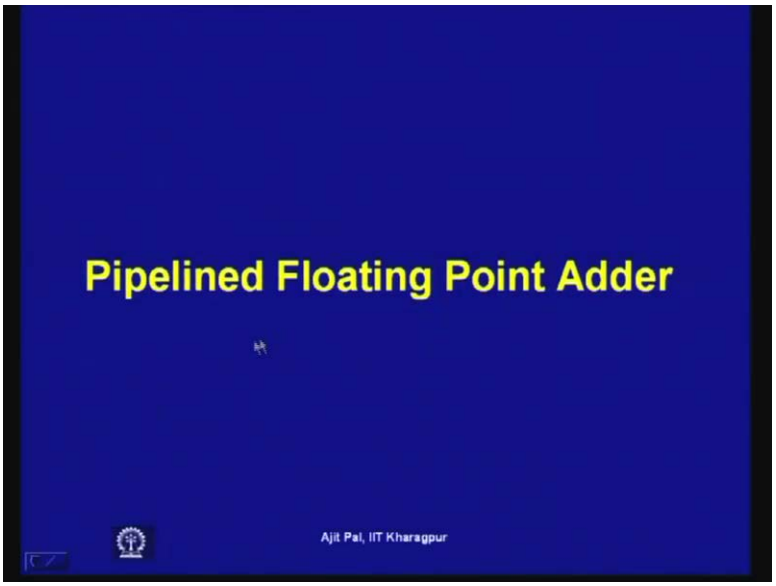
So, this input is going to the second stage and we are applying 3 inputs to each of the carry save adder, they in turn are producing the output. So, you can see the first carry save adder, each is having 10 bit inputs, so it is producing 10 bit output. Similarly, these three inputs are applied with 11, 12 and 13 bits, they will produce 2 outputs, 13 bits each. Similarly, these two are directly going to another stage and the output of this carry save adder are being added or applied to another carry save adder and this is 13, so this is 10, 10, so the most significant bits will be 0s and this will produce 2 outputs.

So, here actually 1 2 3 4 5 6 7 8, 8 inputs are there and ultimately, they are converted into 4 outputs, these four outputs are applied goes to the third stage. And in the third stage, you will require 2 carry save adders to transform them into two 16 bit outputs. So, first three inputs will go to one carry save adder and there are two outputs and the fourth input will go to the another carry save adder and ultimately, you have got two 16 bit outputs.

Now, these two 16 bit outputs will go to a carry propagate adder, which is essentially a ripple carry adder to produce a 16 bit sum and actually, carry output will be also there, which is not shown. So, this is the 16 bit output product, carry is not because here we are interested in the product. So, you get a and we shall assume that, there is no overflow or from this, so we shall be getting a product of 16 bit from the two 8 bit number. So, this is how you can see, in 4 clock cycles, you are able to perform the computation with the help of a number of carry save and carry propagate adder.

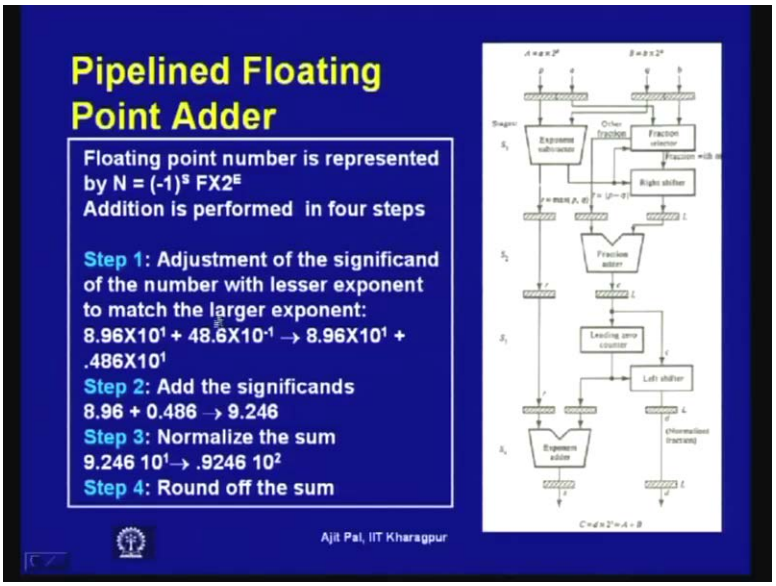
And not only that, if you are have to perform this multiplication continuously, so after the first multiplication is done, which will take 4 clock cycles, the subsequent multiplications will be performed in I mean, in one output will be generated in one clock cycle. So, when you will be performing multiplication continuously, this type of pipeline implementation can be done.

(Refer Slide Time: 51:36)



Now, let us focus on the pipelined floating point adder.

(Refer Slide Time: 51:40)



So, the pipelined floating point adder is also implemented in 4 stages, so before we go to the discuss the implementation of the pipelined floating point adder, let us see what are the operations we normally perform, whenever we go for pipeline addition I mean, normal floating point addition. So, here, we have given an example of decimal floating point number, but it can be binary and the operations will be same. So, here you see, you have got two floating point numbers, one is 8.96 into 10 to the power 1 another number is 48.6 into 10 to the power minus 1.

So, these two numbers are to be added, so first operation that we do is the, adjust the exponent I mean, adjust the number having the smaller exponent and convert it into a number of exponent having the same exponent value, as the larger exponent. That means, for example, this 48.6 into 10 to the power minus 1 is converted into 0.486 into 10 to the power 1, only when this is done then we can perform the addition. So, you can see, we have done, we have shifted this, this can be carried out with the help of shifter.

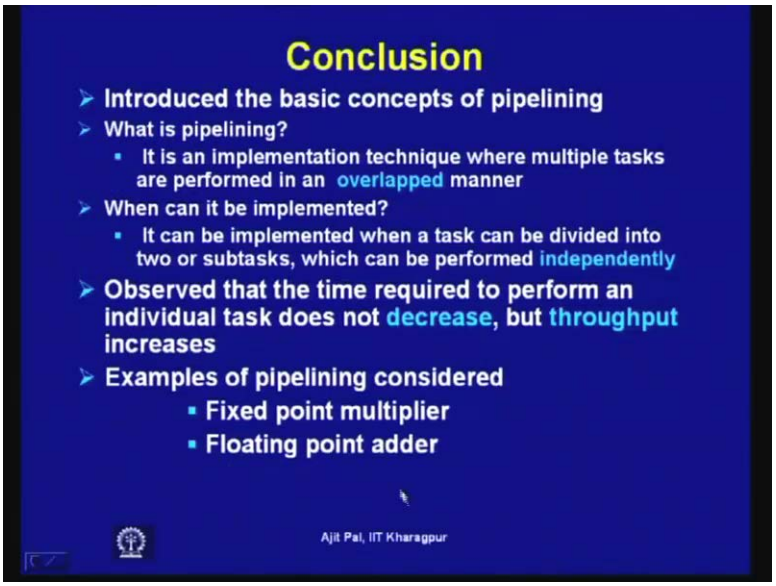
And this adjustment of the significant is done and then you can add the significants, this 8.96 can be added with 0.486 to get 9.246. And then we do a kind of normalization and when we do the normalization, that exponent is adjusted and here, there is no I mean, the number starts with fraction 0.9246 and if there is leading 0s then you have to do the round off. The same thing is done here, you can see exponent subtraction is being done to find out, what is the difference between the exponents.

And accordingly, the fraction with smaller exponent is shifted with the help of right shifter, to have the same exponent value. And that number and the exponent with larger fraction I mean, larger exponent, that fraction these two are added with the help of the fraction adder. And the exponent values are maximum of the two exponents, that is passed on to the third stage. And here, after the addition is done, if it leading 0 in the result, in the fraction, that is being left shifted.

Earlier we did right shifting, now you are doing left shifting to remove the leading 0s and how many shifting is done, that is being counted here and then we get a normalized result, that is passed on to the last stage. However, you have to adjust the exponent value to take care of the number of leading 0s and so exponent is adjusted of the adder and we get the final exponent here and here, the sum we get. So, we get the d that is, the sum of the number and the exponent of the result s is obtained here.

So, this is how that means, d into 2 to the power s, in binary that these are the two number. That means, we starting with two numbers (Refer Time: 55:22) a into 2 to the power p and we are adding with b into 2 to the power q and then it is producing d into 2 to the power, maybe 2 to the power s. So, this is being done in a pipeline manner and this is implemented with the help of a four stage pipeline.

(Refer Slide Time: 55:46)



Now, it is time to conclude, so in this lecture, we have introduce the basic concept of pipelining. You have seen, what is pipelining, what pipeline means and it is essentially an

implementation technique, where multiple tasks are performed in an overlapped manner, after dividing into a number of subtasks. And when can it be implemented, it can be implemented when a task can be divided into two or more subtasks, which can be performed independently, as we have seen.

And we have observed that, the time required to perform an individual task does not decrease, but the throughput decreases, throughput increases. So, whenever we go from non pipeline to pipeline implementation, we have seen, the time required to perform a task does not decrease, if we considered individually. However, the throughput increases, the number of outputs that can be generated per unit time that increases.

And in this lecture, we have discussed about pipeline implementation of fixed point multiplier and floating point adder. These are essentially arithmetic pipeline implementation, but as I have told, the most common application of pipelining is in instruction pipeline. So, in the next lecture, we shall discuss about the pipeline implementation of instructions.

Thank you.