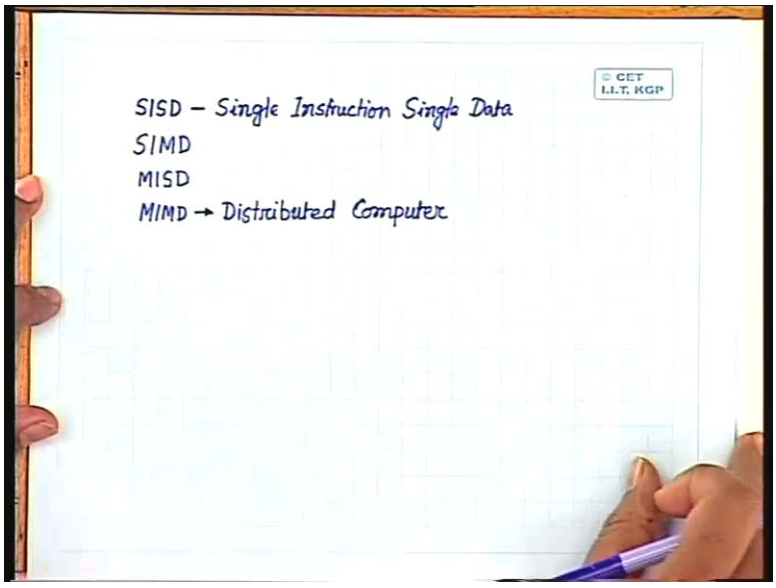
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic & Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 07
Pipeline Concept – 1

Till last class we have seen how to design a CPU. So for designing of a CPU what you have to do is you have to finalize what are the instruction sets or what are the instructions that will be in the instruction set of the CPU. You have to finalize that what are the resources that is how many registers including memory address register, program counters, CAD pointer all these things have to be input within the CPU. We have to decide the data path, a path which will take the data from one resource, one source to another destination within the CPU and accordingly you have to design the control circuit which will control the operation of the CPU, operation of different components of the CPU while execution of a particular instruction. We have seen that the control circuit can be one of two types.

We can have harder control where the control signals are generated by the combinational logic circuit along with some sequence counter and in the other options, we can generate the control signals from a ROM in which case we will call it as a software control or a micro programmed control unit. So along with the data units, the control units whether it is hardware control unit or the software control unit taken together forms the central processing part of a computer. Now the type of CPU that we have discussed till last class that is a class of a machine, a class of CPU which is called a single instruction single data or SISD type of CPU.

(Refer Slide Time: 00:02:51 min)



So SISD or single instruction single data that means the computers built with this kind of CPU is capable of executing only one instruction at a time and that too only one data at a time. So this is a single instruction single data type of computer. Now there are different types of computers

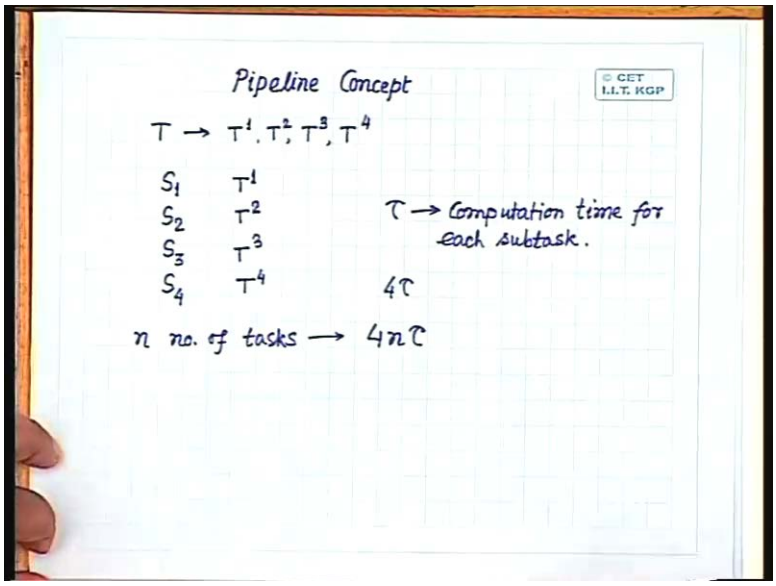
which can be built is single instruction multiple data or SIMD where we will have a number of processing elements. So when you say processing elements, it will consist of few number of registers and ALU but the control unit will be same because it is single instruction multiple data that means I will have a number of processing elements but all the processing elements will execute the same instruction simultaneously but on different data sets.

When a type of architecture which is applicable for let us take for example say image processing kind of applications where if I want to perform some operations say for simplicity, let us assume that we want to add two images. So you know that the images are nothing but a sort of matrix where I can have an image size of say m by n that means it is a matrix of size m by n where every matrix element will represent some intensity value in digitized form. So it is the same operation which has to be performed on every matrix element. So for that kind of application, the kind of architecture which is suitable is SIMD because it is the same instruction, same operation to be performed on different elements. So there the kind of architecture which will be used is SIMD or single instruction multiple data architecture.

We can have other kind of architectures like MISD that is multiple instruction single data and this is most of theoretical importance because getting an application, here applying more than one operation on the same data simultaneously is hardly possible but for completeness, we assume that there is some architecture called MISD or multiple instruction single data but another kind of architecture which is widely used is called MIMD or multiple instruction multiple data type of computer. You will see that this MIMD leads to a type of computing which is called distributed computing.

So essentially what is MIMD architecture is that we can have a number of independent computers. Each of the computer can execute different programs and the different programs will be executed on different data set. May be for coordination, if the different programs are part of the same task then the computers needs to interact while execution on defined data sets and that can be done via local area network or some closely connected network. So different types of configurations are possible. We will come to details on this later on but for the time being let us see and why people have gone for all this different kinds of architectures? It is simply to improve the performance of the computer system. That means we want to do more amount of job in less amount of time. So now in order to do that people have tried for this different kinds of computer architecture.

(Refer Slide Time: 00:07:24 min)

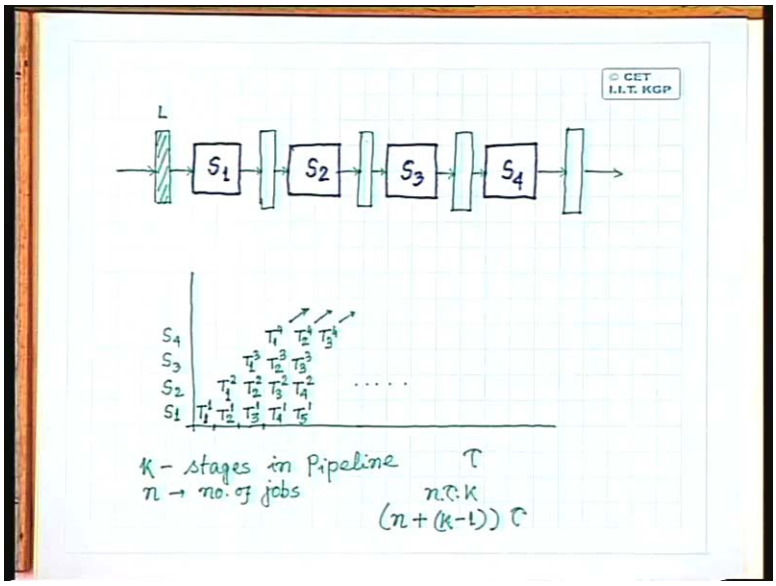


Now let us see a simple concept by using which the performance of a computer can be increased. That is the concept of pipeline. Now what is this pipelining concept? Let me assume that I have been given a task. Let us say task, I call it as task T. Now this task has to be performed by a computer. If it is possible that this task T is divided into a number of subtasks. Suppose I break this task T into a number of subtasks say T_1, T_2, T_3 and T_4. Suppose I break the task T into these 4 subtasks and when I execute, when I complete these 4 subtasks in this given order, the order will be decided by interdependency of the data. So when these 4 subtasks T_1, T_2, T_3 and T_4 are executed in this order then this complete task T is executed. Now if I assume that I have some organization where I have different computing blocks. So I have a computing block say S_1, I have a computing blocks say S_2, I have a computing block say S_3 and I have another computing block say S_4.

Here this computing block S_1 is capable of executing task T_1. This block S_2 is capable of computing task T_2, S_3 is capable of computing task S_3, T_3 and S_4 is capable of computing task T_4. So this is the kind of architecture that I have. I also assume for simplicity that each of these subtasks T_1, T_2, T_3, T_4 each of them takes the same amount of time for computation. I assume that time of computation for each of the subtasks is tau. So this is the computation time for each subtask. Now find that if tau is the time for computation of each of these subtasks then on a single processing machine if I have to compute these task in this given order, then for completion of this task T the time taken is 4 tau because the task T has been divide into 4 subtasks. So that total time of execution for this task T is 4 tau. If I have n number of tasks to be executed by the machine then the completion time of n number of tasks is 4 n tau. This is simple mathematics.

Every subtasks take time tau. I have 4 subtasks for every tasks. So every task will take 4 tau amount of time. For n number of tasks, it will be 4 n tau amount of time. Now let us say another option whether we can improve this total time required. So I have said that I have 4 processing blocks S_1 which is capable of computing T_1, S_2 capable of computing T_2, S_3 capable of computing T_3, S_4 capable of computing T_4 and these tasks are to be executed in order 1, 2, 3, 4.

(Refer Slide Time: 00:12:00 min)



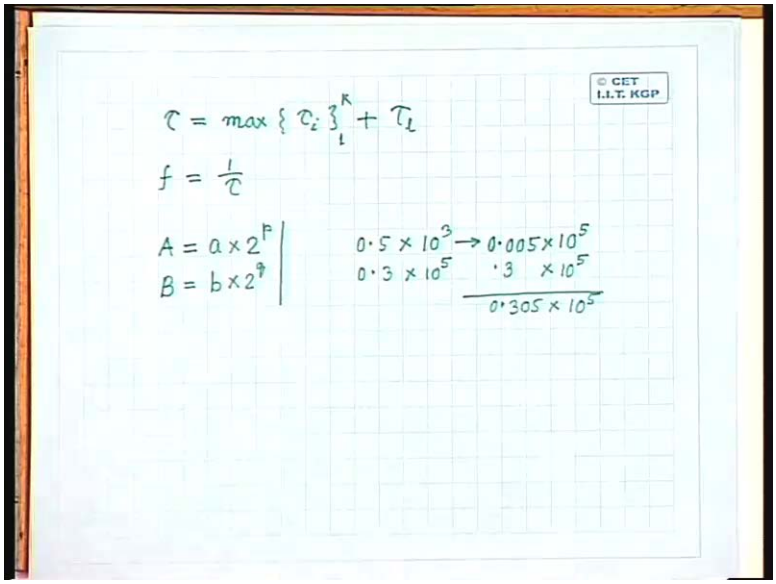
So if I organize these processing blocks in this manner… these are the latches. So whenever a task is to be executed that this is called a pipeline, the tasks is right to the pipeline. Now for a task T, the first subtask that is to be executed is T_1 and this S_1 is capable of executing T_1. So the S_1 executes T_1. Next on the same task, the subtask that is to be executed is T_2 and that is to be executed by S_2. So after completion of subtask T_1, for the same task subtask T_2 is executed by S_2 whereas this first block can be utilized for execution of T_2 of the next task, T_1 of the next task. So if we put this in the form of what is called as phase time diagram, the situation will be something like this. This is called phase time diagram. This is the execution status of S_1 S_2 S_3 and S_4. During the first time interval tau, S_1 is used for executing subtask T_1 of the first task. So the task number, I put it as sub script and the subtask number I put as superscript. So this T_1 1 indicates that this is the first subtask of the first task. So during the first time interval tau, S_1 executes the first subtask of the first task. Then during the next time interval tau because the first subtask is already compared, S_2 can execute the second subtask of the first task.

Simultaneously S_1 can be used to compute the first subtask of the second task. During the next timing interval this one, the third subtask of the first task can be executed by S_3. The second subtask of the second task can be executed by S_2 and first subtask of the third task can be executed by S_1. So this way it can continue. It can continue like this. Now here you find that once the pipeline is full, when I say the pipeline is full when every processing block has got a job to be executed. So when the pipeline is full then you find that after every time interval of tau, I get an output, which if I don't employ such a pipelining concept then after every 4 tau I get an output. So you find that if you have sufficient identical job to be executed and if the jobs can be broken in to a number of such pipeline stages then by applying a pipeline concept like this, I can have tremendous gain in execution of the system, throughput will increase tremendously.

So you can compute that if I have got 5, say k number of pipeline stages, k stages in the pipeline per every stage takes a time tau for execution. If I have n number of jobs to be executed and that case in a non pipeline processer the amount of time that will be taken is n times tau times k. This

is the time taken for execution of n number of jobs on a non-pipelined architecture. Whereas if I have a pipelined architecture then you can easily find out that the time taken will be n plus k minus 1 times tau. So this gives tremendous gain, if I go for a pipelined architecture instead of a non-pipelined architecture. Now if all the pipeline stages takes an amount of time tau, in that case you can easily find out that use of this latches are not essential because whenever one stage completes its execution, the data from the previous stage is ready for passing through the next stages. So if every stage in the pipeline takes the same amount of time tau for execution in that case, we need not have this latches but practically having such a situation that breaking into pipeline stages for every stage, we will take the same amount of time is almost impossible. So in general different pipeline stages will take different amount of time for execution and in which case use of the latches are essential so that the data is not lost. Now in that case we can find out that at which rate we can pump in the tasks. So that will be decided by the stage which takes maximum amount of time. So I can decide tau in that case like this.

(Refer Slide Time: 00:20:53 min)



The tau will be same as maximum of tau_i. The tau_i is the time taken for the execution for ith pipeline stage where i will vary from 1 to k because we have brought k number of stages plus I have to incorporate an additional time because now I have put latches in between. Every latch will incorporate some delay. So if I say that latching delay is tau_L then tau is given by maximum of tau_i where tau_i is the computation time of the i th pipeline stage plus tau_L where tau_L is the latching delay. From this you find that the rate at which the task can be pumped to the pipeline is simply f equal to reciprocal of tau. So this is very simple calculation.

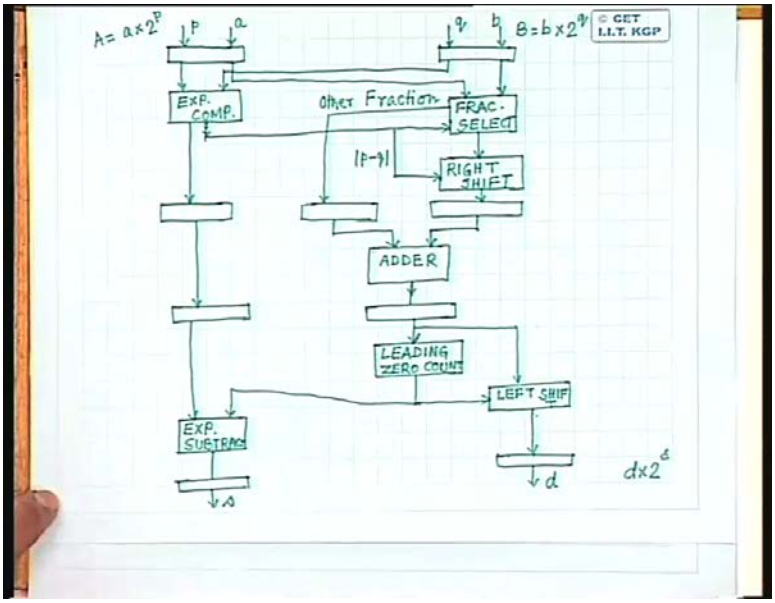
Now let us come to a practical example where this pipelining can be very useful. Say for example we wanted to add 2 floating point numbers. So I want to add 2 floating point numbers say A which is of the form a into 2 to the power p where a is the fraction part, lock as a is the fraction part and p is the exponent part and another number floating point number B which is given by B into 2 to the power q. So these are the 2 floating point numbers that we want to add. Now you know that whenever we wanted to add two floating point numbers, what is the best

way or the most efficient way of doing it. You simply equalize the exponents, after equalizing the exponents you add the fraction parts. So that fraction part along with equalized exponents gives you the addition and if necessary you may have to normalize it. So if I want to equalize the exponent, in that case I have to compare the two exponents of the two numbers to find out what is the difference. Then the fraction of one of the number have to be either divided or multiplied by two or integer powers of two, so that your number remains the same. So simply like this, if I have a one number say 0.5 coming to the decimal domain into say 10 cube. Other number is say 0.3 into 10 to the power 5. What I can do is I can equalize this. This number I will equalize in such a way that my power will become ten to the power 5.

So if I put this as 10 to the power 5, I have to modify this fractions part. How you modify the fraction part? You simply give some right shift to this fractions part and in this case because I have increased the exponent by 2 that means this has to be given right sift by 2 digits. So this fractions part now becomes 0.005 times 10 the power 5, number remains the same, only some adjustment among exponent and the fractions part. Here it is 0.3 into 10 to the power 5 as before. So whenever I want to adjust the exponent and fraction part, I have to know that what is the difference between the exponent, increase the lower exponent by the required amount and give right shift to the fraction part by the required amount. After this is done, your addition is very simple, this becomes 0.305 times 10 to the power 5.

So the result will consist of two components, one is the fractions which is 0.305 and other is an exponent that is 5. In some cases it may be necessary that after performing this addition or subtraction whatever it is, if you find that there are some leading zero's in the fraction part. Normally what we want is the leading zeros have to be eliminated. That means I have to count how many leading zeros after decimal point you have in the fraction part then give left shift to the fraction by so many digits and because you are giving left shift that means fraction you are multiplying by 10 raised to the number by which you have given the left shift. So many times I have to reduce the exponent. So all these operations are to be done, when we want to perform the addition or subtraction of the floating point numbers. Now let us say that if I want to perform similar type of task through a pipeline processer, what are the pipe line stages that we meet.

(Refer Slide Time: 00:26:36 min)



So the first component in the pipeline stage will obviously be 2 latches which will latch the fraction and the exponent. So I have two latches, one latch will take the exponent part and fraction part of the first number. So exponent part p and fraction part a because we have said that we have to add number A equal to a in 2 to power p. This is the first number. The second number that we want to add is number B which is b in to 2 to the power q. So this will also take exponent q and fraction b as input. Then the first operation that we have to do is compare these 2 exponents p and q and this will be required for normalization purpose. So I must have an element which is able to compare the exponents. So this is exponent comparator. Exponent comparator will take the exponent p and exponent q and it gives some output. Once I find out that which of the exponent is greater or which of the exponent is lower, I have to select to one of the fractions, one out of this A and B for normalization.

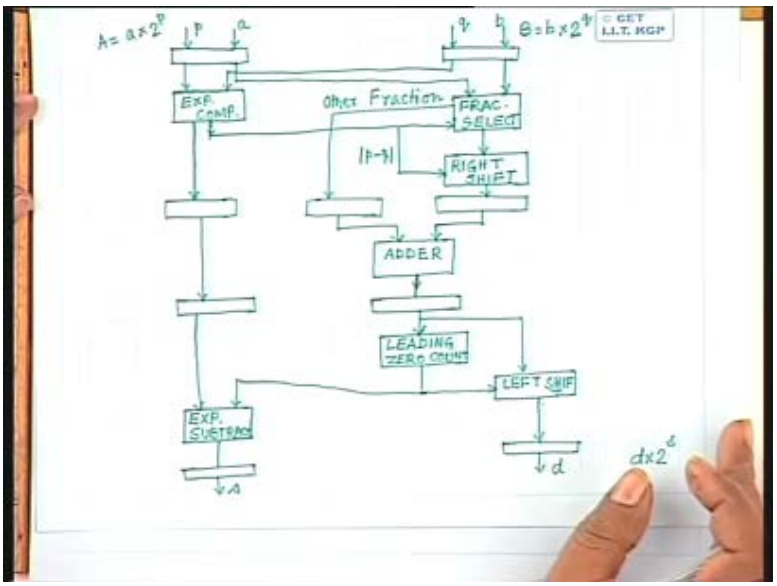
So on this side I have to have a fraction selected. Fraction selectors will take this fraction inputs B and A, it will get a signal from the exponent comparator to tell you that which of the fractions is to be selected. Now once you select a fraction then I have to give required number of right shift to the selected fraction. So I have to have a right shifter. Is not it? Simply by this example I have taken this fraction 0.5 and given right shift to this by 2 digits. So I have to select to one of the fractions, give the required number of right shift to that selected fractions. So this will get an input from the fraction selector and at the same time, I have to know that how many digits are in this case how many bits of right shift I have to give. That depends upon what is the difference between p and q.

So from this same exponent comparator, I get a signal which will indicate that how many bits of right shift is needed and this is equal to given by subtraction difference of p and q. So many bits of right shift has to be given to the selected fraction and after that what I need is, so following this I can put another latch.

So you see that output of this right shift is the normalized fraction. I have to add this normalized fraction with the other fraction which is not normalized. So I have to have other fraction coming to this latch. So this gives you the other fraction and here from the exponent comparator, I get the maximum exponent. Now after performing this, I need one adder element that is the fraction adder which will add the original fraction with the normalized fraction to give you the resultant fraction. This resultant fraction I can latch in to another one and the exponent that is selected. You find that the number of latches that you need in both this paths have to be same otherwise there will be time delay and may lead to data loss. So the number of latches that I have in this path have to be same as the number of latches or the stage of latches that I need in this form. So this gives you the resultant fraction.

After resultant fraction typically what we need is leading zero counter for renormalization of the resultant fraction. So here I have to have a leading zero counter. Then the same one depending upon the number of leading zeros you have, I have to give left shift to the resultant fraction and number of bits by which the data has to be left shifted that depends upon how many leading zeros you have. So this is the block which gives left shift and by whatever number of bits you have given left shift to this, this exponent has to be incremented by the same number. Isn't it? Because if I give some number of left shift to this that means I am multiplying by the number. If I give one bit left shift that means I am multiplying the fractions by a factor 2. Two left shift means multiplying by a factor 4. So by which ever amount, by which ever number you give left shift, the same number has to be detected from the exponent because one multiplication has already been done here. So to keep the numbers same, the same number has to be detected from the exponent. So what you have to do is we must have an exponent subtractive or which can also be done by adder because you know that subtraction can also be performed by adder.

(Refer Slide Time: 38:11)



So I will put it as exponent subtractor and this exponent subtractor has to get the output from this leading zero counter. The exponent will be subtracted from the selected exponent and then the finally what you get is the result.

So I will put one more latch here. So here I get the final result and the final result in the form, suppose this is the exponent and this is your fraction part d. So your final result will be simply d into 2 to the power s. This can be a total pipe line architecture for addition of two floating point numbers. Now what is the advantage that you get by using such a pipeline architecture? If I have an application where I simply have to add 2 numbers, just 2 numbers I have to add to get a result. Does this pipe line architecture give any advantage? Possibly no. The pipe line architecture gives you an advantage, when you have to do identical operations on a large set of data. So for example if I have two linear arrays of floating points numbers, arrays a and b and these to be arrays are to be added to generate an array c, again of floating point numbers. The array may be of size of 100 or 500 or 1000. That means I can consider that each of the array use a vector, per every component of the vector is the floating point number. So two such vectors I have to add to generate a third vector. In such kind of application, this type of array architecture is very useful. So first you feed in the first component of the vector.

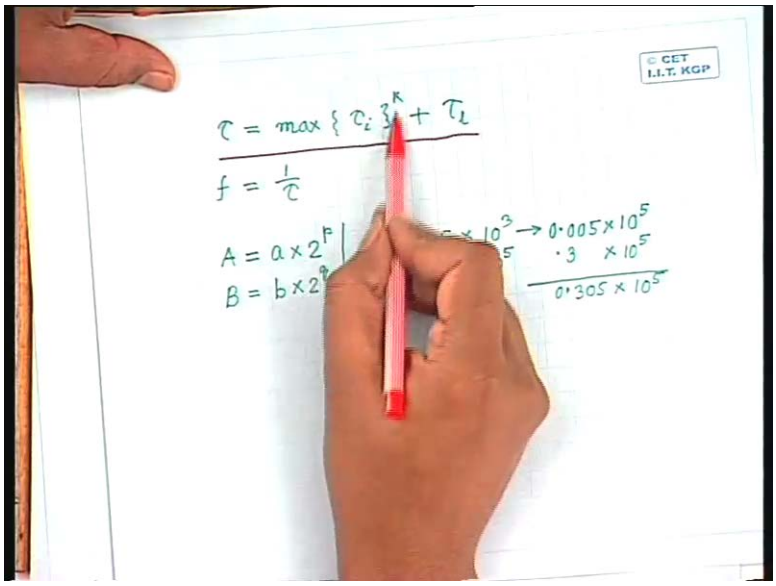
After computation of the first stage in the pipeline, the first component comes to the next step. When the first stage can be used to compute the first pipeline function of the second components. So this way if you proceed, you find that this pipeline architecture gives tremendous advantage over non pipeline architecture and the amount of gain that you can get we have already said else this much. For a non-pipeline architecture you have n times tau times scale. For a pipeline architecture, this multiplication is reduced to an addition that is n plus k minus 1 times tau. This gives tremendous advantage, when you have identical operations to be performed on a large volume of data and that is what leads to the concept of vector computer or vector processer.

So in case of vector processor the type of application that you need, suppose I wanted to add two arrays [Conversation between professor and student - Refer Slide Time: 00:38:37 min] [no, they are not that is why the latches are input in between]. So what we have to decide is at what frequency or at what rate the pipeline should work. [Conversation between professor and student - Refer Slide Time: 00:38:52] [No, number of latches in this. See here I have two distinct paths]. I can simply demark it between two parts. This side deals with the exponent, this side deals with the fraction. Number of stages I have to have on the exponent side has to be same as number of stages, I have to have on the fraction side. Otherwise the amount of delay that we incorporate on this side will be different from the amount of delay that we incorporate on this side and because of this delay mismatch, it is possible that you get only the fraction. You don't get the exponent or you get only the exponent but you don't get the fractions.

So to avoid that you have to ensure that whatever delay you encounter here, the same should be and encountered here. There can be delay mismatch between different stages. So here this is one stage of pipeline, this is another stage of pipeline. May be this can further be subdivided, another number of pipelined stages that you have, more number of pipelined stages more advantage we will get. Your speed gain is more, if you have more number of stages in the pipeline. So this can be broken into further pipeline stages that is also possible. Different pipeline stages may take different amount of time. So this will be a more advantageous, if we can come close to the ideal. Now what is close to the ideal that every pipeline stage will take some amount of time for computation. If I can guarantee that, in that case some of these latches I can avert. Of course on this side I cannot avert because here there is no computations, simply passing the data from one latch to another. But assuming that in every stage I have some computations and every stage

takes same amount of time for computation. In that case the latches are not consistent but if the time taken for different stages of the pipeline is different. In that case it is necessary that I have to put a latch otherwise the data will be lost. If you want to put a latch in that case we have to decide that at which frequency the pipeline has to operate and this is the equation for it.

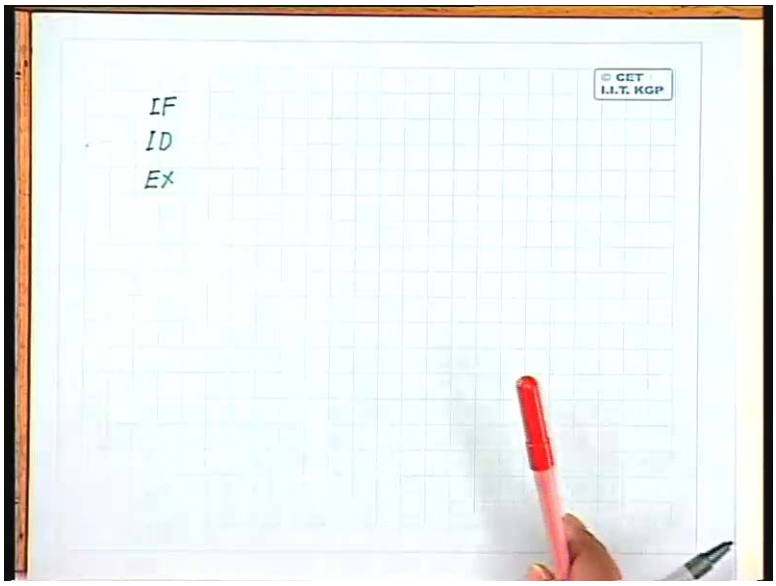
(Refer Slide Time: 00:41:28 min)



So you find that the pipe line frequency, the rate at which the pipeline has to operate depends upon the stage which takes maximum amount of time and that is what is called pipeline bottleneck. So it is the block which decides maximum at which rate the data can flow. It is simply the concept of the bottle at which rate you can take out the fluid from the bottle that depends upon what is the width of the neck of the bottle. So that is why the concept came, the naming came, it is the pipelined bottle. What do you mean by identical? (Refer Slide Time: 00:42:33) delays have to be the same but see ideally what we assume is all these are synchronous circuits, all the latches. They operate only either at the (Refer Slide Time: 00:42:50 min) falling edge, they are all latched. There are various such examples in which you can have search pipeline computations. Now this is a type of pipe lining which is called execution pipelining.

Now let us see that how the simple pipelining concept can be expanded to the cpu design. We have said that in case of any cpu execution of any instruction consist of a number of cycles. The first cycle is an instruction fetch cycle, I call it IF. The second operations that is to be performed on this instruction which has been fetched is instruction decode. So I call it instruction decode or ID.

(Refer Slide Time: 00:43:41 min)



After the instruction is decoded than only the instruction can be the executed, so I call it the instruction execution cycle. Now you find that when instruction is fetched, the cpu makes use of the external address box, external data box, memory all these things, the control box. When the instruction is decoded, in that case those things are not needed. You don't need this box, you don't need that data box, you don't need to access the memory.

Similarly when instruction is executed, the instruction decoder is not needed. Address box, data box, memory control box all those things are needed optionally depending upon what is the instruction that is going to be executed. If it is memory read or memory write instruction, in that case all those external boxes, address box, data box and memory they will be used. If it is something with in the cpu, say for example transferring a data from one register to another register in that case none of these external boxes are needed. So it was thought that why don't you make use of these boxes to prefetch some instructions when they are not really needed for execution of the already fetched instructions. That means essentially what we want to have is over lapping of these different stages and different instructions in time which will lead to a pipeline concept. So that we will see in the next class.