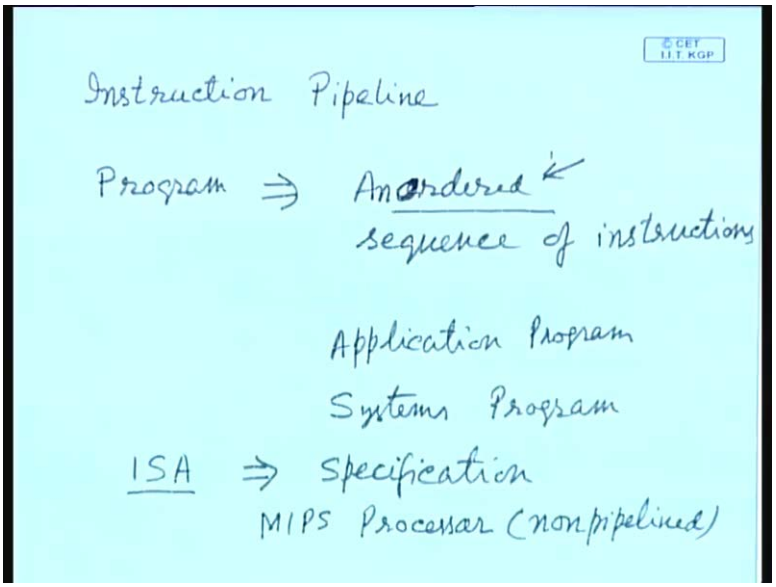
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 7
Instruction Pipeline

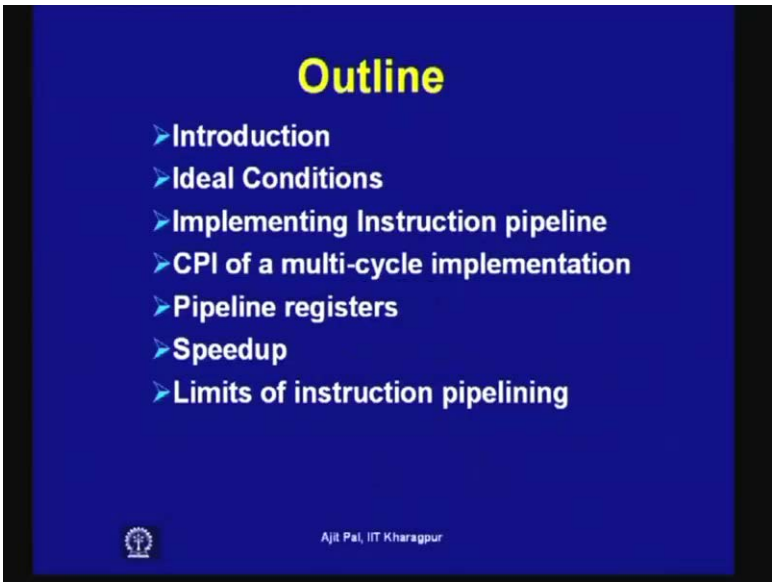
Hello and welcome to today's lecture on Instruction Pipeline. In the last lecture I have discussed in detail the basic concepts of pipelining. What is pipelining? When can it be implemented and how can it be implemented, in general terms I have discussed that, and if you remember I told that pipelining can be implemented for a task. If a particular task is repeated large number of times. And if you look the processor I mean the way the processor works a computer processor of computer works, you will find that as soon as you turn the power on it is starts executing instructions.

(Refer Slide Time: 01:53)



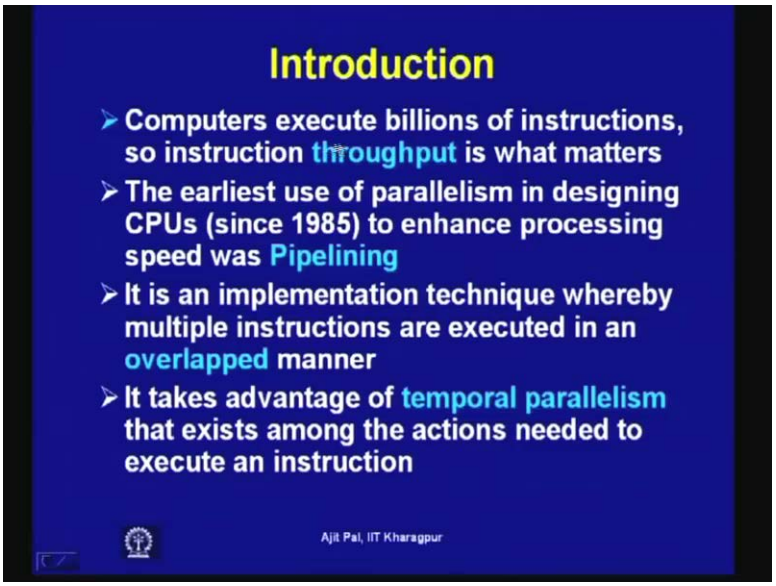
Because basic job of a computer is to execute program, and program is nothing but a or say an order sequence of instructions. And whether it is a application program or a system program respective of that you will find that a program is nothing but an order sequence of instructions. And obviously, the you have to later on you will understand the significant of this of the meaning of this order, why order this say sorry, because you cannot really arbitrarily n write a sequence of instructions, they should be in some ordering depending in the application you are implementing.

(Refer Slide Time: 03:19)



So, introduction pipelining is very important and particular we will see that instruction pipeline is being used of since a 80's in processors it is implemented since 80's. And here is the outline of today's lecture on instruction pipeline, after giving of to introduction I shall talk about ideal conditions. That means, the ideal conditions for instruction pipeline, then how the instruction pipeline can be implemented, I shall discuss about CPI of a multi cycle implementation. Then pipeline registers need for pipeline registers, then speedup achieve by using instruction pipelining. And of course, there is some limits on instruction pipeline that also I shall discuss in this lecture.

(Refer Slide Time: 04:08)

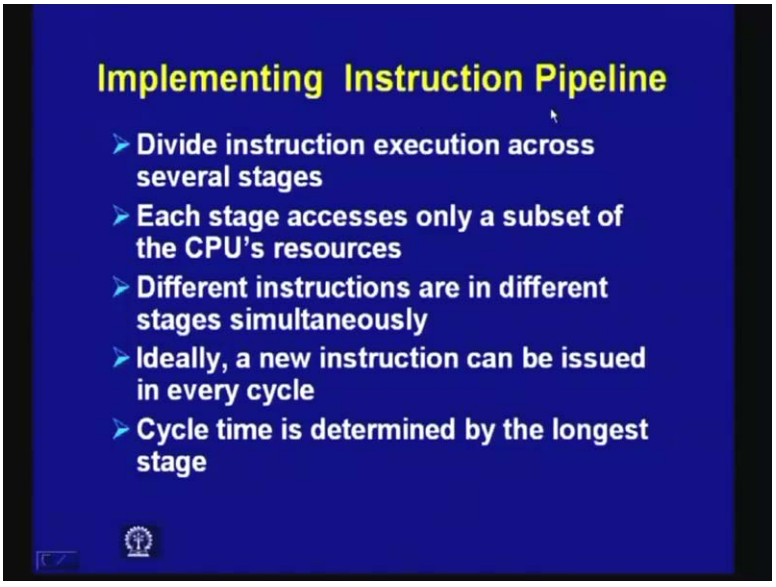


As I have already mention computer execute billions of instruction, so instruction throughput is what matters. To improve the performance of the processor, instruction throughput is very important, throughput of processor when for executing the instruction is very important. And that is reason why a instruction pipelining has been used for a long time, and this is the first kind of parallelism that was incorporated in processors since 1980's.

And the earliest I mention since 1984 pipelining was used to enhance the processor speed. And it is and I have already mention that pipelining is nothing but an implementation technique; that means, you can have a an instruction set architecture (Refer Time: 05:00) ISA, Instruction Set Architecture represents the specification. And that specification can be implemented in many ways, a processor with the same ISA can be non pipeline, a processor with the same ISA can be pipeline.

So, the instruction set will not change, so for as the users view programmer view the concern it will not change. But, there will be difference in the execution time, there will be speedup and so on. So, I without effecting the ISA implementation is done and in the last lecture I have discussed about the MIPS processor ISA of MIPS processor. And discuss the non pipeline implementation of MIPS processor, and today I shall extended for pipelining implementation.

(Refer Slide Time: 06:23)



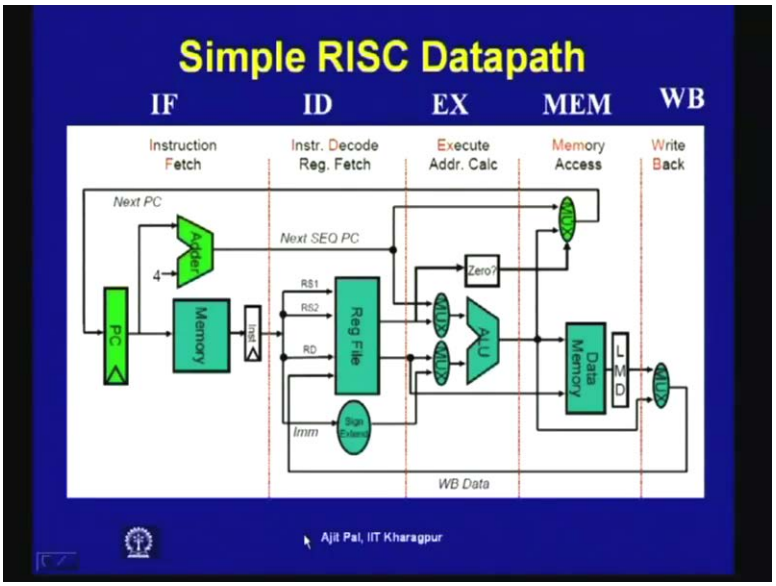
How can it be done, so to implementing the instruction pipeline these are the following steps are to be performed, number one is divide instruction execution across several stages. So, what you have to do here the task is execution of an instruction, and that task has to be divided in number of subtasks, as I mention pipelining can be implemented if task can be divided into more than one subtasks. So, the execution of the instruction should be divided into several subtasks that is one very important requirement.

Second is, each CPU accesses only a subset of the CPU's resources, a Central Processor Unit will have various recourses, like Arithmetical Logic Unit, adder, registers multiplexers, bus and so on. So, a subset of these recourses will be used for a executing a particular subtask as rather subtask of instruction, and different instructions are in different stages simultaneously.

Here as you have seen in our earlier example, whenever you are executing several tasks in a pipeline manner, different subtask of different task will be in different stages of execution. Similarly, here also you will find that subtask of different instructions will be in different stages of executions, when you go for implementing instruction pipeline. And ideally a new instruction can be issued in every cycle, as you have already seen in the earlier case per cycle one task is entering the system.

And here also we shall try to I mean issue I mean one instruction will be entering the pipeline, when you go for executing a instruction. But, I have used term in the beginning ideal; that means, ideally this is the possibility, but in real life you will see that we may have to deviate from this ideal condition. And cycle time is determined by the longest stage as I have already discussed at length, in the last lecture you may have different sub different stages. And different stages may take different amount of time to perform the different subtasks, and with a longest subtask the stage which is taking longest time, will decide the clock frequency of the pipeline system, so these are the basic implementation issues.

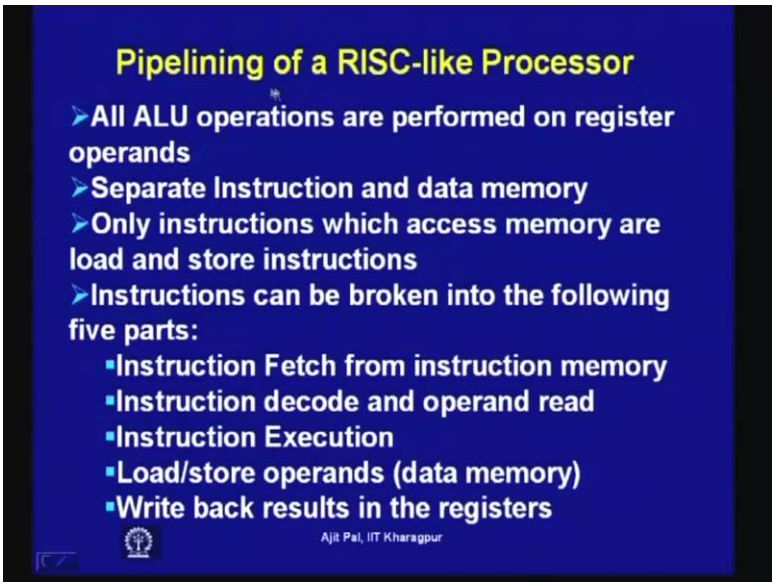
(Refer Slide Time: 09:40)



Now, coming to simple RISC data path which is consider the non pipeline data path, we have already discussed it at length. This is that MIPS data path, which has got first stage is performing instruction fetch, second is stage performing instruction decode, and register fetch, third stage is performing execution, fourth stage performing memory access, fifth stage is performing write back.

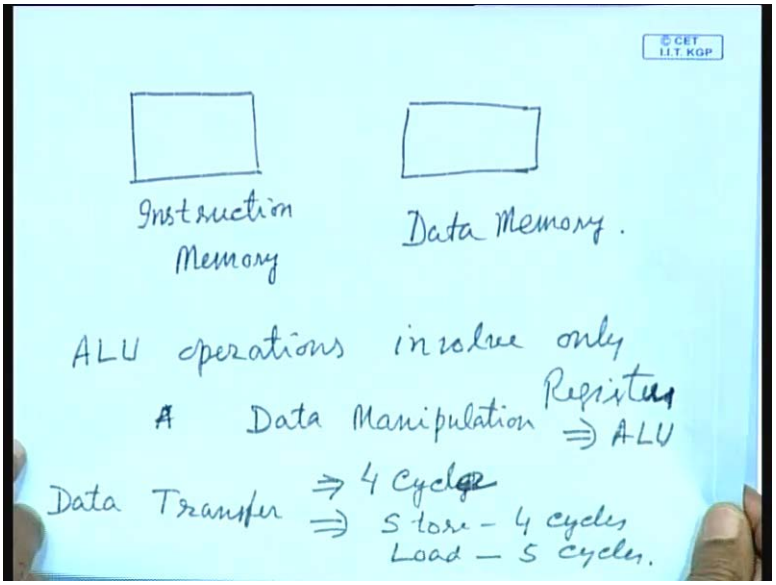
So, we got five different stages and these are the five different, you know operations which are been performed while executing the instructions. So, it is natural for us to implement stage such that each of these operations is performed in each stage, so we shall be implementing pipeline with five different stages.

(Refer Slide Time: 10:50)



And particularly, we have taken of the RISC like processor for implementation of pipeline. Because, of various advantages of RISC like processor or RISC processor number one is all ALU operations are performed on register operands, I have already highlighted in detail. The difference between RISC and SICS processor, and particular the RISC processors are simpler in terms of for implementation, and these are the key features again highlighted all ALU operations are performed on register operands. and separate instructions and data memory.

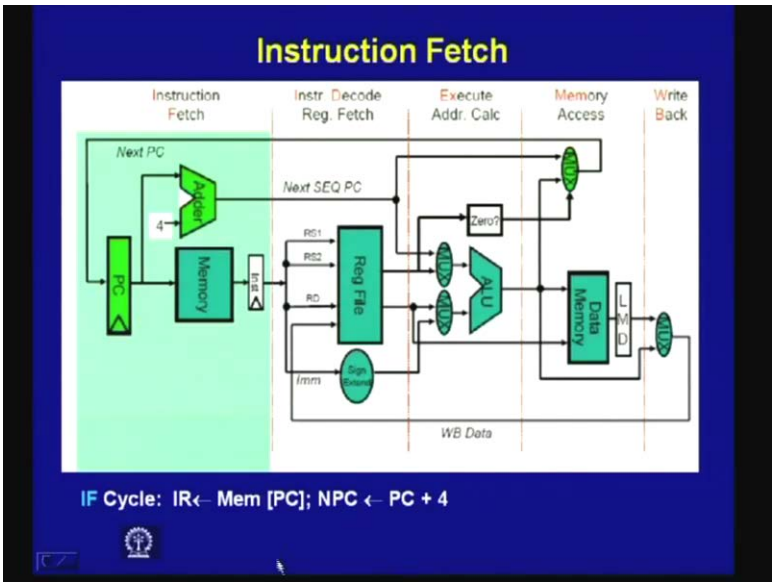
(Refer Slide Time: 11:44)



So, we shall be using two separate memories, instruction memory where program should be stored. And another memory that will be used is data memory, and later on we shall see used of these two memory systems in single system will help in implementing pipelining, it will facilitated easy implementation of pipelining. Only instruction which access memory are load and store instructions, so load and store instruction are the only instructions which will access memory.

Because, we have seen that ALU operations involve only registers; that means, operands will be taken from registers, results should stored in the registers for all arithmetical logical operations. And as I have already mention instruction can be broken into the following part, instruction fetch from instruction memory, instruction decode and operand read, instruction execution, load and store operands and write back results in the registers.

(Refer Slide Time: 13:21)

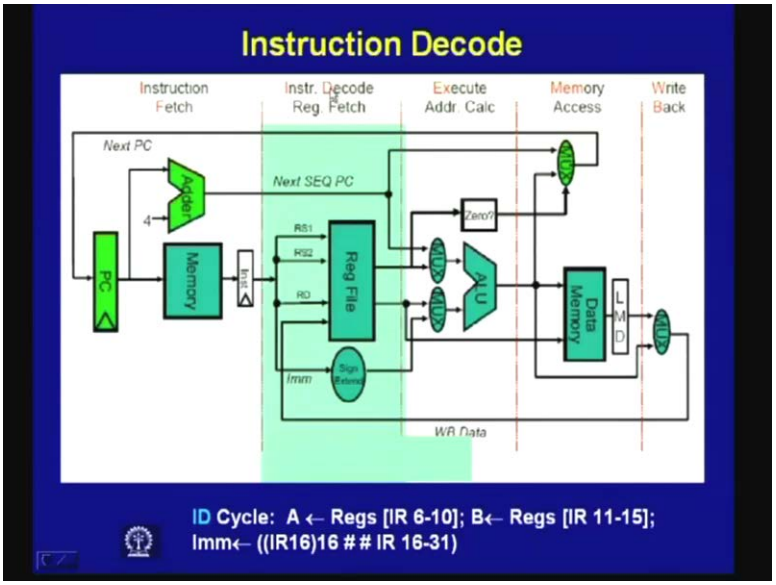


And to highlight again the operation of different cycles, there is it you can see left hand side that is instruction fetch is said in shaded form. And so the operation that is preformed in the instruction fetch cycle is you are loading the instruction register, you have got a instruction register. And that instruction is still you are loading by reading the instruction from the memory, by the address supplied by the program counter.

So, program counter is giving you the address, and that address is used to fetch the instruction from the memory, and that is been loaded in the instruction register. In

addition to that, it is also performing the calculation of the next program counter value by adding 4 to the present value of the program counter. So, NPC is equal to program counter plus 4 that operation is also perform in this instruction fetch stage you can say.

(Refer Slide Time: 14:30)



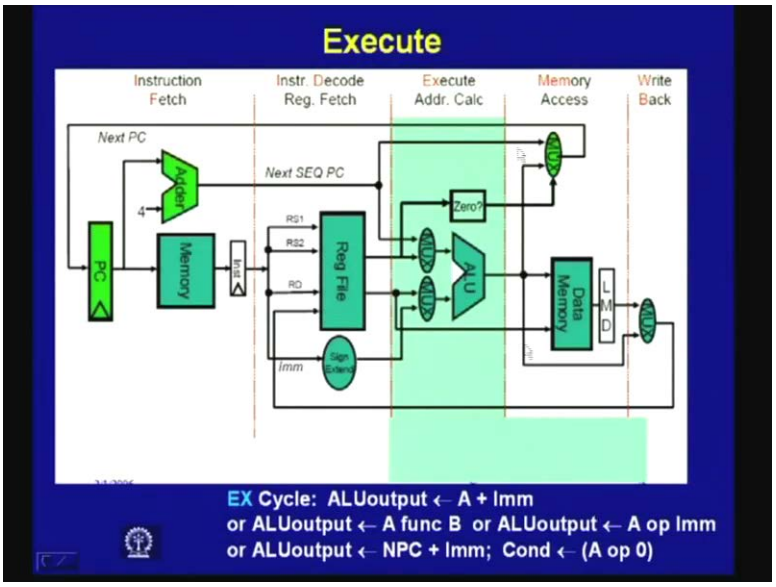
Then the next stage your instruction decode, instruction decode will take the input from this register instruction register, and it will apply that instruction register will provide input to the register file. So, you can see operands they will come from a particular register and the field 6 to 10 will provide you the address of operand 1. Operand B will be taken another register, and the field 15 to 16 will provide you the address of the register.

So, these two are the operand addresses A and B and that will apply to the register, and also in this particular stage. You will calculate the immediate data you know that address value, by adding the immediate 16 bit immediate value data is available as part of the instruction and this is sign extended generate 32 bit data in this stage also. So, of course, it will depend what kind of instruction it is executing, so if it is a you know all ALU operation; obviously, this is will not be required. But, whenever load store instructions are perform then this operation is require.

However, the hardware is there for both I mean fetch reading register as well as for generating the immediate data, sign extension of the immediate data. So, this is the

instruction decode, so this can be the second stage of the pipeline in our pipeline implementation of the processor.

(Refer Slide Time: 16:18)



Then the third stage will perform the execution, so third in the execution cycle, depending on the instruction that is getting executed it will perform different things. For example, one possibility is that ALU output will be equal to A value of A that is been applied here, and the immediate data that is been applied here. So, these two will be added to generate an address, so generate some result with the help of this third stage address.

Actually this will be immediate for in load and store instructions, where the address is generated in this manner. Then it can perform different arithmetical logical operations and depending on that, the that type of instruction ALU output will be equal to A function B; that means, depending the operation to be perform addition, subtraction, multiplication, then and or so two operands are available here. And that operation is been perform bit by bit operation or addition or subtraction whatever it may be, and result is produced here.

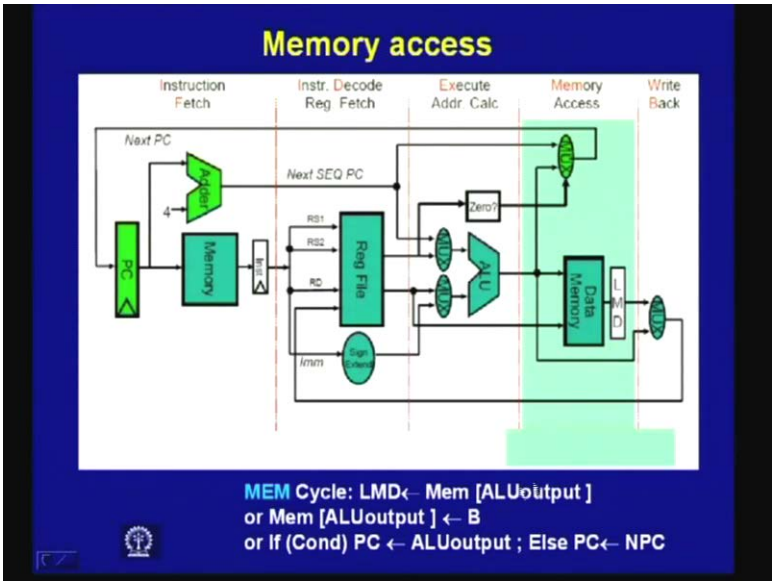
Then ALU output is equal to A operation immediate, so whenever you are performing using immediate mode of addressing. Then it will perform this operation A value of A will be applied to one hour of ALU, another ALU provided with this immediate data, and

that they used to be added and it will produce the result here. Rather not added it can be any operation provided by the ALU control signals.

Then the ALU output next one is your NPC plus immediate, so sometimes it is a used to generate the address. So, NPC which is applied here with that immediate data will be added to generate the address, which has to be subsequently loaded in the program counter. So, this type of things will be required in branch, jump this type of instructions.

So, and of course, this will be dependent on some conditions, so condition is decided here that condition value 0 calculation is also done. That means, to whether the result is 0 that computation is done, and depending on that this multiplexer output is selected whether it will go from this branch address or it will be taken from the PC plus 4, so this is also done in stage 3.

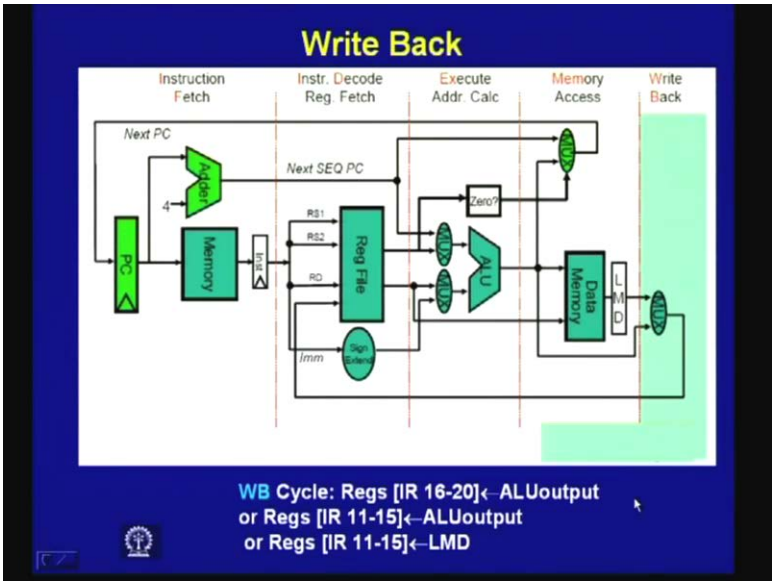
(Refer Slide Time: 19:04)



Then for load and store instruction you will require the memory access, so that is performing in that can be done in stage 4. Where you will load there is a register load memory data, which will be loaded by the value coming from the memory, and address will be provided by the ALU. So, ALU output is giving you the memory address, and that output is applied to the memory, and the data is been stored in this load memory address register or it can be memory ALU output.

So, the value of B value I mean B will be loaded in memory location, so in this particular case, register file will provide the data. And address is supplied by the ALU, and in that is your load instruction, where the data will be loaded into the memory and these are the conditional cases, if condition then value output loaded into the program counter, ALU output is loaded into the program counter, this way LPC is equal to next PC. So, you can see these are the it will this stage 4 will perform all the memory operation needed for different types of instructions.

(Refer Slide Time: 20:26)

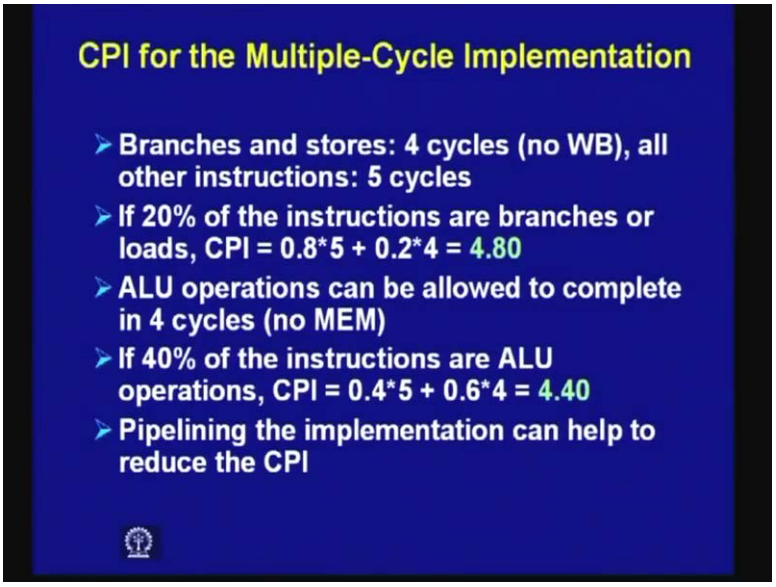


And that in a last stage is a write back stage, in the write back cycle the particular register has to be loaded, write back mean you are essentially writing back the result into the register. So, the multiplexer output is providing in the value to be stored and it will go to the register, the address will be supplied by the instruction itself, so that is the field supplied by the by the instruction field 16 to 20 that will give you the address, and data will be coming from the output of the multiplexer.

And that value will be loaded in the proper register or it can be that ALU output can be directly also loaded, in some instructions or that the data coming from the load memory data that in case of store, this memory data has to be loaded into the register. So, sorry that is your load other one was stores, storing means you are storing the value from the register into memory that is store. And for whenever you are loading it from the memory to the register it is load.

So, load and store in all these cases this write cycle is write back operation is required, so you can see we have divided the ALU operation in 5 stage, identified the function to be perform by different stages. And also we have identified the necessary hardware resources that you required in for different stages. So, in this way you can form the different stages and implement the pipeline system.

(Refer Slide Time: 22:20)



Now, before we go for it is pipeline implementation, let us see some kind of comparison of this I mean before we can implement pipelining, and compare with existing non multi cycle implementation. Let us see, what is the value of CPI in multiple cycle implementation, and also later on we shall see what is the value of CPI in case of pipelining implementation. As you can see here, as you know we have got different types of operations (Refer Time: 23:01) like data manipulation.

The data manipulations are essentially the ALU operations, in our pipeline in this particular case. If you go for a multi cycle implementation that is been one cycle for this, one cycle for this, one cycle for this, one cycle for this and one cycle for this, how many cycles are needed to perform different arithmetical logical operations. You can see all over arithmetical logical operations involve only that register, so you do not require data to read from the memory.

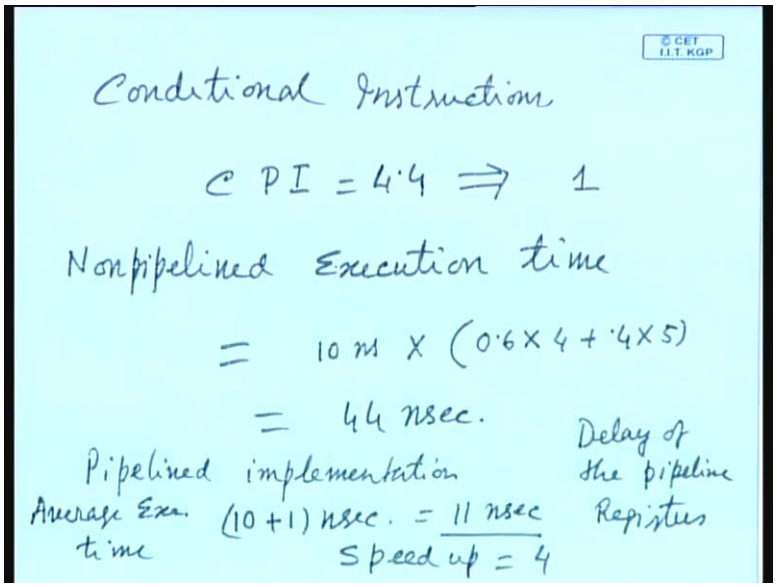
So, it will require first cycle for fetching instruction, second cycle is require for instruction decode, third cycle is require for execution. Because, memory read is not

involve; however, you will require another cycle to ride the result back into the register. So, you require 4 cycles, (Refer Time: 24:18) you can skip the you memory cycle, so instead of 5 cycle you require 4 cycle for all data manipulation.

Now, what about the data transfer, the data transfer operations involves transferring data from the memory to register or from register to memory, how many cycles it involve. So, as you can see that from the whenever you are performing store, then you will require only 4 cycles. Because, the data will come from the register and at this generated in the third cycle, in the forth cycle you can perform the writing operation; that means, your store will require 4 cycles.

However, load will require 5 cycles, why load will require 5 cycles because you can see here in the fourth cycle the address will be calculated, and only in the fourth cycle you will read the data, and fifth cycle you will able to write the result into the register. So, 5 cycles will be required, so we find that if you go for multi cycle implementation then 4 or 5 cycles are required.

(Refer Slide Time: 26:13)

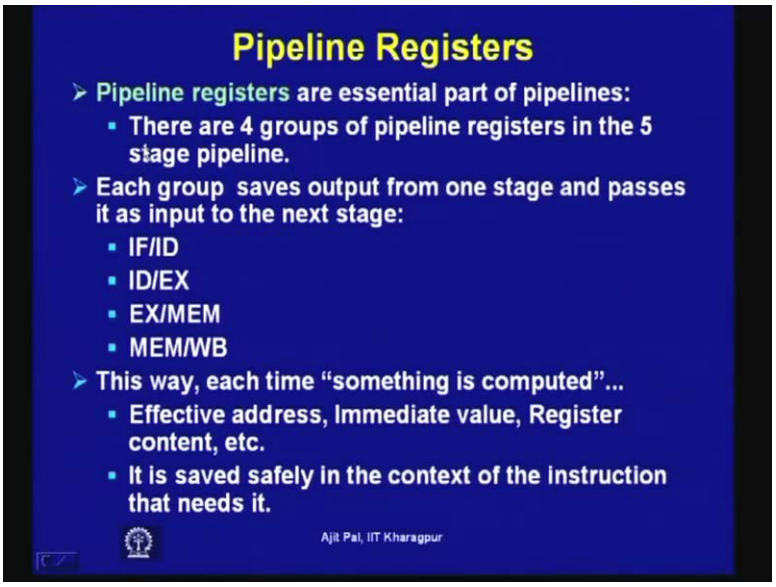


Now, comes to the conditional instructions they are also you will require either 4 or 5 cycles, depending on whether you have to get the address from the memory. Whenever a branch is taken and stored the result into the result from that I mean you have to jump to that particular remain location. So, we find that either 4 cycles or 5 cycles are required for different types of instructions.

And we have done some computations here branches and stores 4 cycles all other instructions 5 cycles, if this assumption is made. Then CPI becomes equal to 0.8 into 5 plus 0.2 into 4 because 80 percentage of instruction will require 5 cycles 20 percentage only 4 cycles 4.8. However as I have already told ALU operation can be allowed to complete in 4 cycles, in such a case they break up will be 40 percent of instruction are ALU operations, 20 percent are branch and stores and so you left with 40 percent which will be require 5 cycles.

So, 0.4 into 5 and 0.6 into 4, so that gives you a CIP of 4.4, so you are getting a cycle per instruction is 4.4. Now, what is the objective of pipelining, pipelining the implementation can help reduce CPI or objective is to reduce the value of cycle per instruction, so by whenever you shall go for pipelining you will see that this will be reduce to 1 instead of 4.4 it will be reduce to 1.

(Refer Slide Time: 28:18)



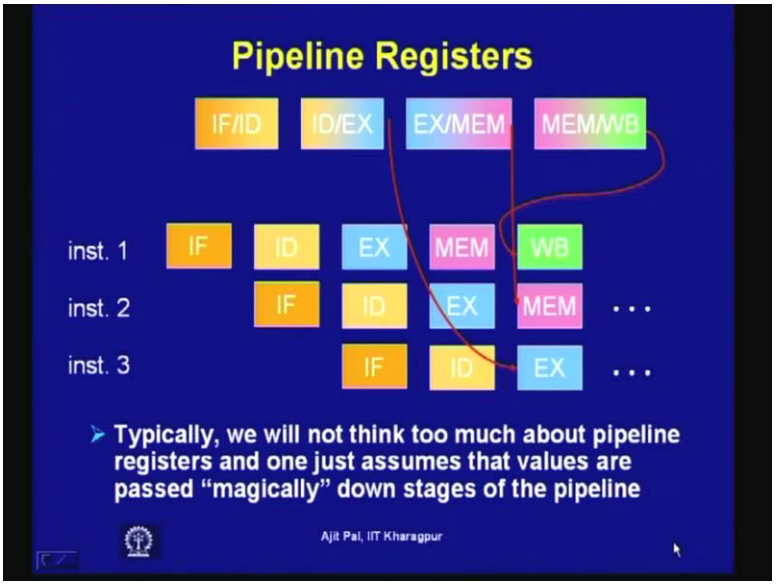
Now, one very important requirement for pipelining is pipeline registers, we have already discussed about the need for pipeline registers. Pipeline registers are essential part of all pipelines, and there are 4 groups of pipeline registers in the 5 stage pipeline, for our pipeline I mean for our data path, which you are interested in pipelining we require 4 stages of memory. Each groups saves output from 1 stage, and passes it as input to the next stages.

So, one register stage will be between instruction fetch and instruction decode that is why the name is instruction fetch, slays IF slash ID. Second stage is ID slash EX, third is EX slays MEM memory, forth is MEM slash WB, so you require 4 such different blocks of registers for your pipeline implementation So, this way each time something is computed that something can be generate in generation of effective address, generation of immediate value, generation of register content etcetera.

So, these are computed by different stages and they will be stored in the registers, so it is saved safely in the context of the instruction that needs it. So, you may be wondering why only 4 such stages of register files are required why not file, so let us look at the pipeline, at each line red line you require one register file one here, one here, one here and one here find out here in the beginning.

The reason for that is that program counter is actually surfing the job for the register for that stage. That means, that instruction fetch stage is getting it is input from the program counter, so program counter is providing the necessary information for stage 1. So, you do not require a separate register file for stage 1; however, for the remaining stages you required separate registers.

(Refer Slide Time: 30:59)



Now, let us see how different how the pipeline registers are used in whenever you go for go on executing different instructions. So, this is the instruction fetch stage, and this is the pipeline registers, these are the additional registers that you require apart from the

register that is present in the ALU. So, instruction fetch slash instruction decode, so this is at the interface of instruction fetch and instruction decodes stage.

So, the instruction fetch stage will perform fetching of instruction from the memory, and it will stored the result into this instruction fetch instruction decode register. And then as you go to the next cycle, you can see the output of the instruction fetch stage instruction decode stage will provide the necessary input, to the instruction decode stage that will correspond to the instruction 1. And at that time you will see a second stage the first stage will be performing instruction fetch.

So, first stage is performing instruction fetch and the result that was produced by instruction fetch is now available in that pipeline register, which is now applied to the instruction decoder. And in a next cycle what will happen that instruction decodes stage will perform necessary decoding, and that instruction decode stage will put the result in that instruction decode instruction execution register. And similarly the instruction fetch register, instruction fetch stage of the second instruction will go to the instruction fetch instruction decode register.

If you go to the third cycle, we can see the output of the instruction fetch is going to instruction decode for second instruction. On the other hand, the output of the instruction execution register pipeline register are applied to the execution, for the that corresponds to the fir first instruction. So, you can see that information about the different instructions are being stored in this pipeline registers, and they are used properly for performing parallel operations.

So, here you can see in the third cycle, instruction fetch is going on for the instruction 3, instruction decode is going on with the inputs coming from this pipeline register. And similarly execution of the first instruction is going on with inputs coming from this pipeline register, ID slays EX. And in the forth cycle in the end of the third cycle, the results produced by the 3 stages are again stored in this 3 registers.

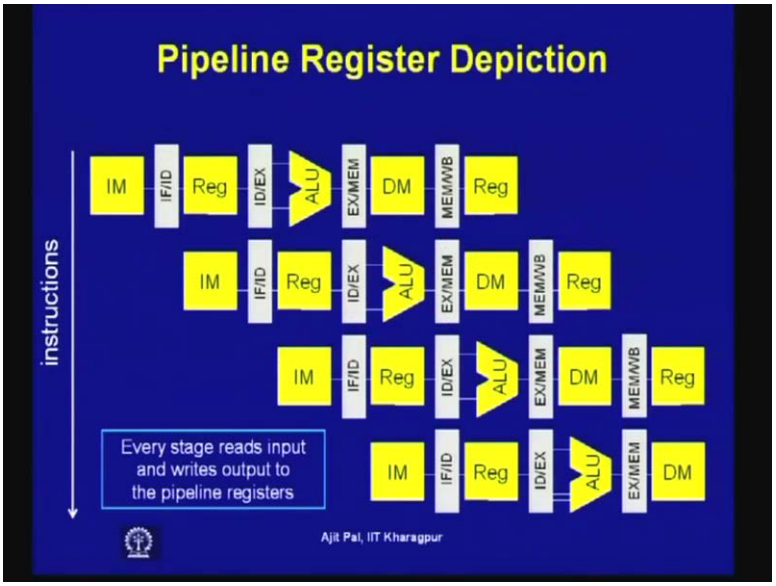
And in the next cycle you are getting output from the registers, and going to memory the corresponding to the forth cycle the that execution, and memory pipeline registers will provide the output, which can be stored in the memory. And similarly that is for the first instruction, for the second instruction that instruction decode was done, so output was

stored in this register pipeline register. And which will provide the input to the execution stage for the next instruction, instruction 2.

Similarly, for the third instruction, instruction fetch was completed and that was that is now available in the pipeline register, and that is being applied to the instruction decode. So, in this way this is will continue and continue the execution, and typically we will not think too much about the pipeline registers and one just assumes that values are passed magically down stages of pipeline or all I am trying to tell you at this point, you see pipeline registers are present.

So, I have explain it in detail how the pipeline registers are being used to save the intermediate results produced by different stages in different cycles, but subsequently we shall not bother about it. So, we shall assume that magically the information is passing from one stage to another and the result is been generated, and parallel execution of different stages of I mean different instruction are getting executed in their different stages.

(Refer Slide Time: 35:56)

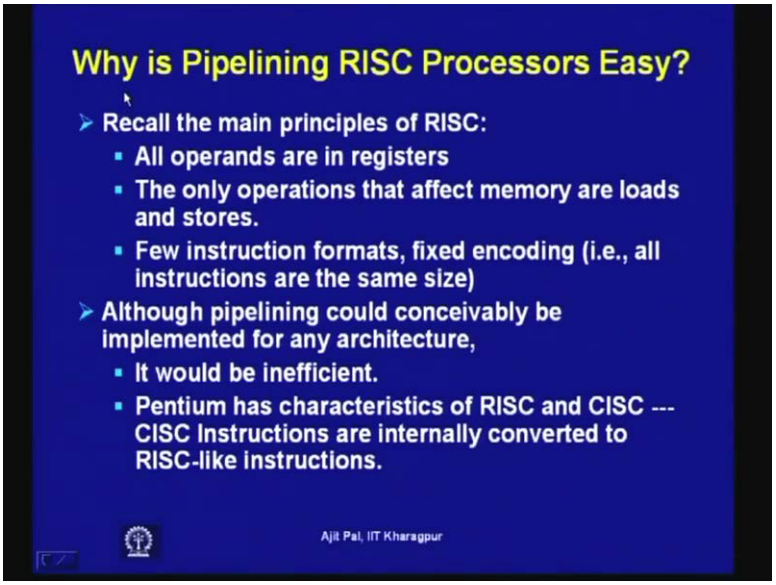


So, this is the pipeline registers and you can depict the pipeline registers in this way you can see here, different instructions this is the top one corresponds to instruction one. Next line corresponds to instruction 2, next line corresponds to instruction 3, the next line corresponds to instruction 4. So, 4 different instructions the resources that is being used are depicted for different instructions, but as you can see here, if you are consider a

particular instant of time you will be finding that resources are different instructions are using different resources at a particular instant of time.

For example, if you consider say first, second, third and forth cycle in a forth cycle this is the situation. That means, the write back is going on sorry in the forth cycle data you are that memory operations are going on, so memory resources getting used in addition to the that pipeline register. And for instruction 2, for instruction 2 that ALU resources being used and along with the pipeline register ID EX, for the third instruction it is using the pipeline register instruction fetches instruction decode. And also it is using the register resource and so for the forth instruction is concern it is performing instruction fetch. So, instruction memory being used, so here you can see that instruction memory and data memory, both this resources are used simultaneously by different instructions.

(Refer Slide Time: 37:56)

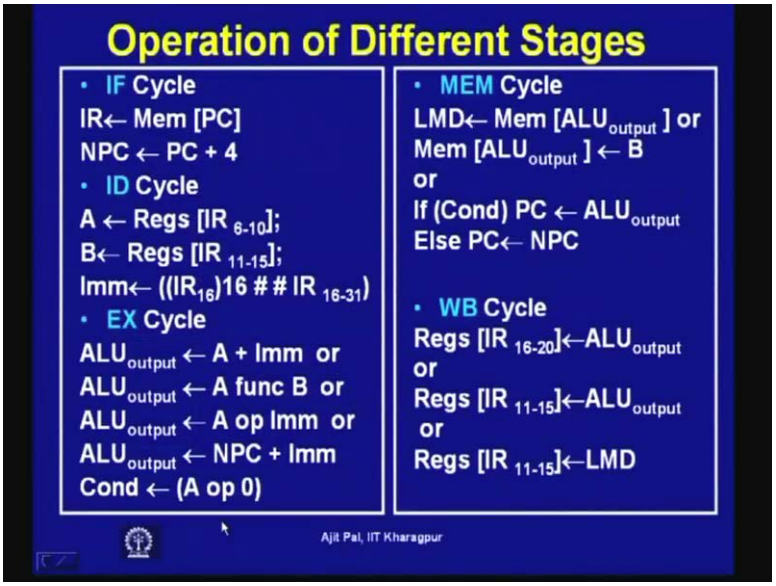


Now, you may be wondering why is pipelining RISC processors easy, I have already explained that all operands are registers. And if that not in the registers then implementing pipeline will be difficult, because when executing instructions you have to fetched the operands from the memory. So, that will incorporate come more complication and that is the result why CISC processor implementation of pipeline for CISC processor is rather difficult. But, it has to be done later on we shall consider the pipeline implementation of say Pentium, and how can it be done we shall see later.

Then the only operations that affect memory are loads and stores I have already mention about it. So, although pipelining code conceivably be implemented for and architecture, it would be inefficient; that means, for CISC processor it will be inefficient, Pentium of characteristics of CISC or RISC. Actually Pentium belongs to the CISC category, CISC instructions are internally converted to RISC like instructions, so this is a just hint.

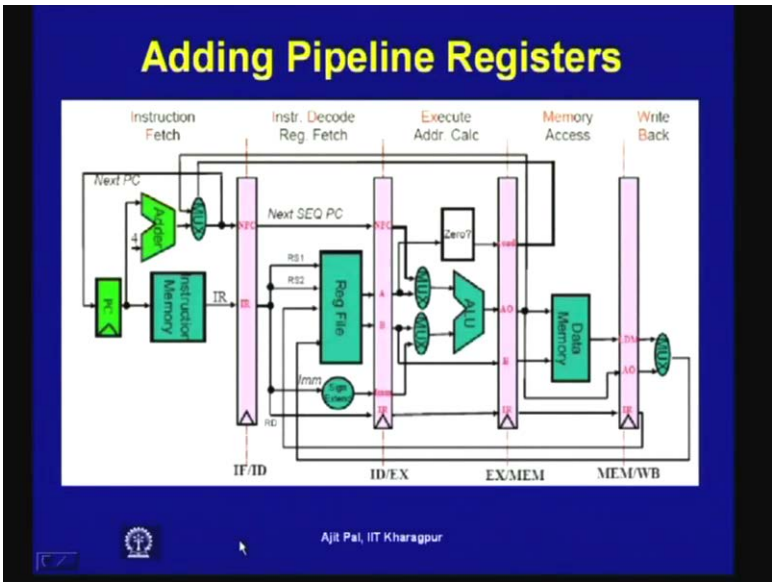
You will see that two implement pipelining, internally complex instructions are converted into RISC like micro operations, then pipelining is implemented. So, directly the instructions cannot be pipeline, but you will require some hardware which will convert complex RISC like instructions into a several simple RISC like operations then they can be pipeline.

(Refer Slide Time: 40:02)



So, that we shall discuss later on for the time bind we satisfied with this observation.

(Refer Slide Time: 40:08)



And this I have already discussed in detail operation of different stages, and now here after incorporating the different pipeline registers, this will be the look. That means, this is the first stage, stage 1 that is your instruction fetch stage, and this is the instruction decode and register fetch. And in between that instruction fetch instruction decode register pipeline registers have been incorporated.

Similarly, between that instruction decode register fetch, and execution stage instruction decode register fetch stage, and execution stage we have put another register pipeline register that is your IS slash EX. Similarly between the execute stage and memory access stage, another pipeline registers have been incorporate that is your EX slash MEM register file. And finally, between the memory access and write back, another registers stage have been incorporated that is your memory slash WB.

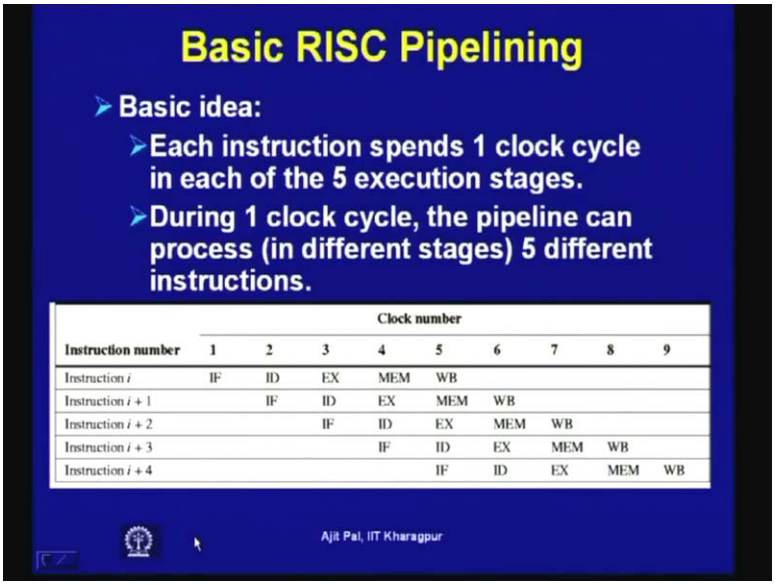
Now, you can see we have not only we have added some registers, we have remove some registers that also you have to noticed. For example, the non pipeline implementation if you go back, you will find that we had some registers like instruction registers, there was an instruction register, there was a load memory data register. Those registers are no longer required because these registers the function of this registers are being implemented with the help of these registers for though.

For example, that instruction register is no longer required because this particular register instruction, fetch slash instruction decode this register is actually holding the instruction

for the next stage. So, that instruction register is no longer required which stage required in non pipeline implementation, similarly at the end of this data memory there was a load memory data register, in a non pipeline implementation that is not required. Because, this particular register pipeline register memory slash write back MEM slash WB, this particular pipeline register will hold the data coming out from the data memory.

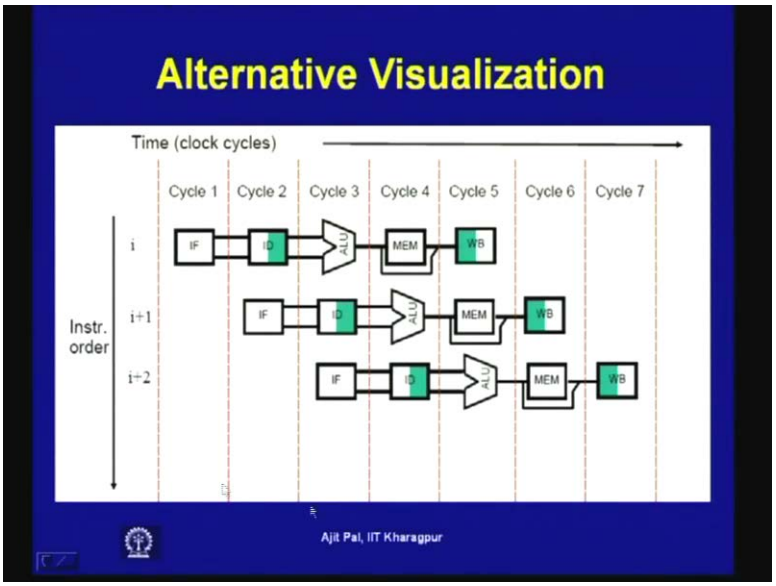
And that will provided to multiplexer for storing it to the register or in a subsequent cycle. So, we can remove few registers, but; obviously, we have to incorporate more complex, and larger number of registers to implement the pipelining.

(Refer Slide Time: 43:19)



So, whenever you implement this pipelining then the basic idea is each instruction spends one clock cycle in each of this 5 execution stage, based on our you know that ideal condition. We have we have assumed that all these stages will take same time, and they are as a consequence, the clock cycle required is one, so the each instruction spends 1 clock cycle in each of these 5 execution stages. And during 1 clock cycle, the pipeline can process 5 different instructions which can be depicted in this manner as well.

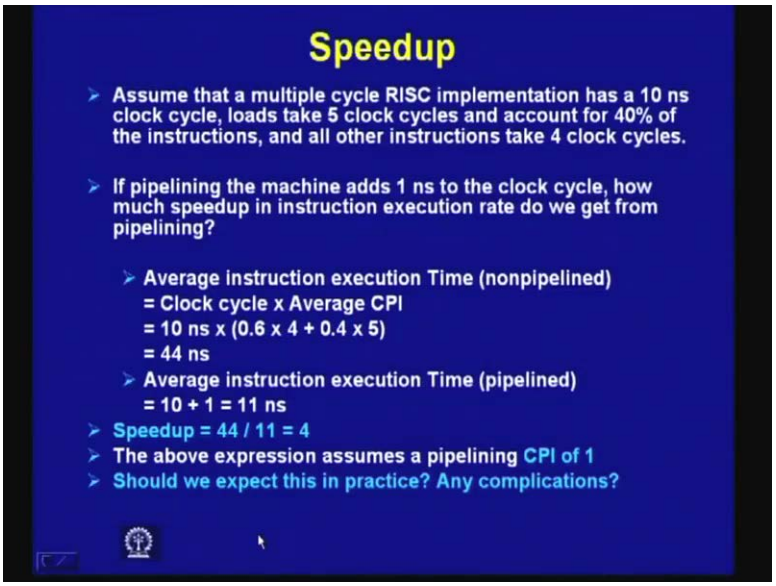
(Refer Slide Time: 44:03)



So, we have seen the different ways of depicting it, this is one visualization for the purpose of visualizing the pipeline execution, either you can visualize in this manner or you can visualize it in this manner. So, both are used in different situations, so this corresponds to instruction 1, next line corresponds to instruction 2, third line corresponds to instruction 3 or to generalize it i plus 2 that is the first one is i, second one is i plus 1, third one is i plus 2 and so on.

And this depiction where you are telling clock number at the top 1, 2, 3, 4 and so on, and then you are writing down the name of the different stages. Alternatively you can use this, where also you have got different instructions in order that is i plus 1, i plus 2 and you can show the different blocks, different stages, instruction fetch, instruction decode ALU memory write back and so on. And clock cycles are also mentioned at the top, clock cycle 1, clock cycle 2, clock cycle 3, clock cycle 4 and so on, so this is alternative visualization.

(Refer Slide Time: 45:22)



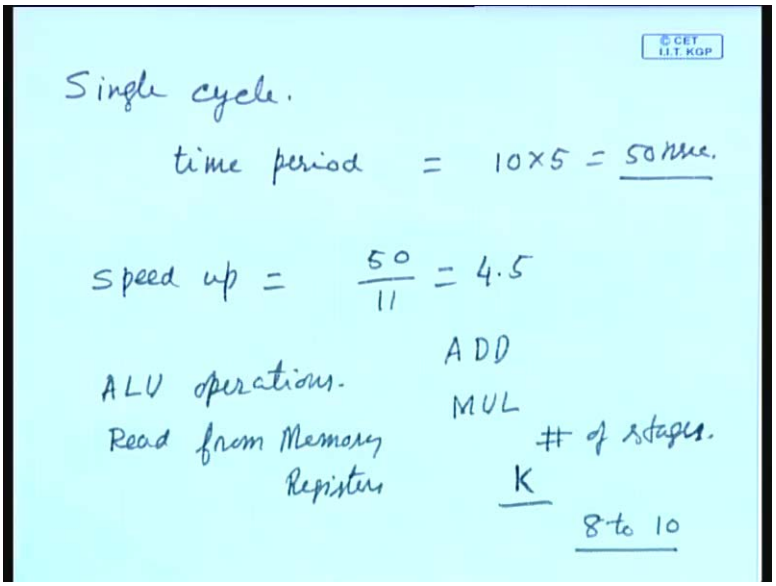
Now, coming to speed up, so assume that a multiple cycle RISC implementation has a 10 nanosecond clock cycle, loads take 5 clock cycles, account for 40 percent of the instruction. And all other instructions take 4 cycles I have already explained this in detail, and that is how we got a CPI of 4.4. Now, only thing that has been added here, the cycle time has been given here that is equal to 10 nanosecond. So, whenever you consider the average instruction execution time, average instruction execution time for non pipeline will be equal to 10 nanosecond into CPI.

CPI as you have already seen that is equal to 0.6 into 4 because 60 percent of the instruction require 4 cycles, and 40 percent of the instruction will require 5 cycles. So, this gives you 3.4 and this will be equal to 4.4 when her 2.4 plus 2 that is 4.4 into 10 that is you 44 nanosecond. And pipeline implementation how much time it will take, so here one assumption has been made in pipeline implementation, add 1 nanosecond to the clock cycle. Why you are adding one nanosecond to the clock cycle because you have to take into the account the delay of the pipeline registers.

So, pipeline registers will involve assumption delay, so it will require 10 nanosecond plus 1 nanosecond that is your 11 nanosecond. And 11 nanosecond is the cycle time for it is execution time for each instruction, I mean that is the rate at which it will come and as consequence. So, we can say that average execution time in case of pipeline implementation is 11 nanosecond, so therefore, speed up is equal to 4.

So, if we consider pipeline and non pipeline implementation, non pipeline multi cycle, now instead of multi cycle if we consider single cycle in that case what is the speedup.

(Refer Slide Time: 49:02)



So, for single cycle in case of single cycle that clock time period of the clock has to be 10 into 5 that is your 50 nanosecond. So, 10 into 5 because whenever you go for multi cycle implementation, your average execution time will be reduce, but if it is single cycle then; obviously, your average execution time will be longer, because the total delay for different stages here for taking to account to decide the clock frequency, so 50 nanosecond.

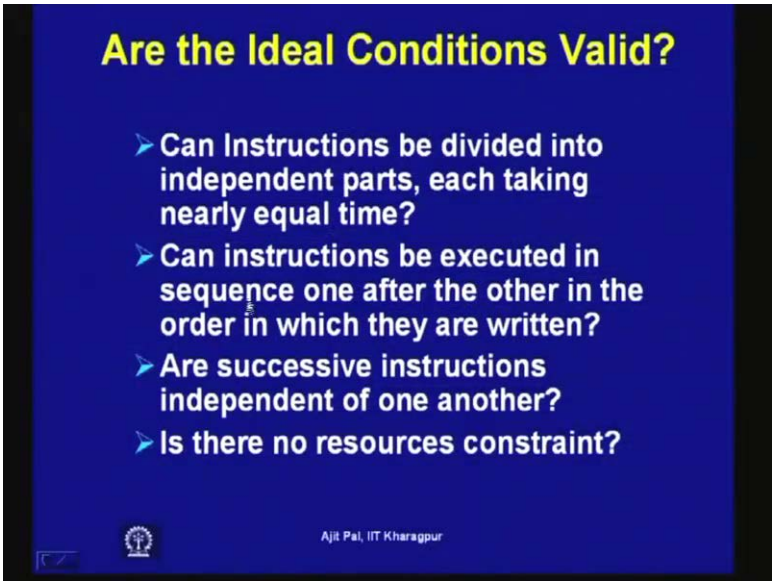
So, in that case the speedup will be equal to 50 by 11, so here it is a more than 4, so 4. point something 4.5 roughly. Now, so the above expression assumes a CPI of 1, here we have assumed that the pipeline processor always generates 1 output per cycle, should we expect this in practice any complications here is a question. We have assumed some ideal conditions, and based on that ideal conditions it is possible to have CPI is equal to 1, so what are the ideal conditions.

So, these were the ideal conditions, we assumed that all instructions are divided in independent parts, each taking a equally a nearly equal time. Now, the question is can instructions be divided into independent parts, each taking nearly equal time that is not true. Because, the whenever you are performing read operation from register, it will take much shorter time, compare to reading data from memory. Similarly, whenever you are

performing add operation, time will be much less compare to whenever you perform multiply operation.

So, even when you are performing ALU operations, different instructions will require different time that ALU operation execution time that time will be different. Similarly, as I said read from memory and read from registers their time will also be different, so in practice that cannot be true.

(Refer Slide Time: 52:11)



Then another question another ideal condition was related to this question can instructions be executed in sequence one after the other in the order in which they are written. So, all I am trying to tell is in order execution as I said a program is nothing but an order sequence of the instructions, the order in which they are present in the program they will be executed in the same order that assumption will also not be true.

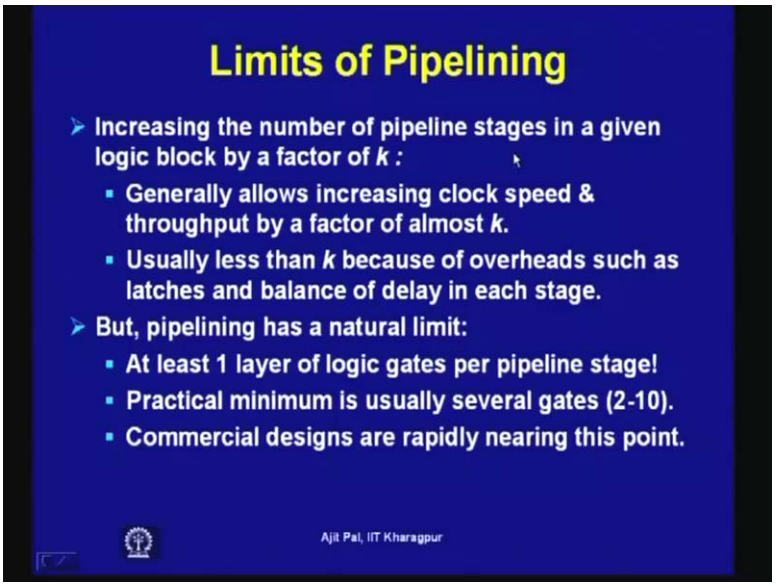
Later on we will see that we shall go for some specialized process I mean some processors. Where there will be some kind of predictions or speculations, where to improve the performance of the processor you have to allowed out of order execution. So, in such situations this will not be valid, third is are successive instructions independent of one another, this is a very common question we assumed that the instructions are independent.

So, they can be executed in a overlap manner, since they are independent there is no problem, but in reality that is not, so. There will be various kind of dependences data dependence, control dependence and so on, so because of these dependences they cannot they are not really fully independent, so these assumptions are not valid. Last part not the least is there no resources constraint; that means, we assumed that there is no resources constraint unlimited resources are available.

But, in practice it is necessary to impose some restriction on the resources to reduce the cost of the implementation. So, the resources are to be optimally used and whenever we are not when these ideally conditions are not satisfied, then your pipelining we shall not get CPI of 1 that CPI of 1. Because, we have to add some babul or it some additional time will be required, some time will be wastes, some clock cycle should be wasted and we shall deviate from this CPI of 1. So, our objective is to get CPI of 1, but because of these problems we shall not get CPI of 1.

Another last point that we would like to mention is limits on pipelining, we have seen the speedup is related to number of stages K. And in an ideal situation, when you are executing large number of instructions, this speedup is K; obviously, we will be tempted to increases the number of stages. Instead of 5 why not go for a 10 stages, so that we get a speedup of 10 or why not 100 stages, so that we get a speedup of 100, but in practice then that cannot be done. Because we have seen that primary requirement is that you have to divide an instruction, into different parts and you have to implement them by using hardware.

(Refer Slide Time: 56:08)



So, increasing the number of pipeline stages in a given logic block by a factor of k, generally allows increasing clock speed and throughput by a factor of almost k as I have already mention. Now, usually less than k because of overheads such as latches and balance of delay in each stage, so we do not get the exactly k as we have seen we got 4.4 or something like that. But, pipelining has a natural limit, natural limit is at least one layer of logic gates per pipeline stage, you see you will be implementing the pipeline with the help of hardware.

And that hardware will require at least one logic stage, usually 8 to 10 logic stages are present in a single stage, but the limit is at least 1 or usually 2. So, that will put a limit on the maximum number of stages that you can have, and practical minimum is usually several gates 2 to 10. And of course, commercial designs are rapidly nearing to this point; that means, in the commercial designs you will find they are trying to increase the number of pipeline stages to as many as possible. So, that get, so that you get higher speed up. So, with this riddles come to an end of this lecture, and in my next lecture we shall discuss about those non ideal conditions, and the impact of those non ideal conditions.

Thank you.