Digital Systems Design
Prof. D. Roychoudhury
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 34
Design of Computer Instruction Set and the CPU (Contd.)

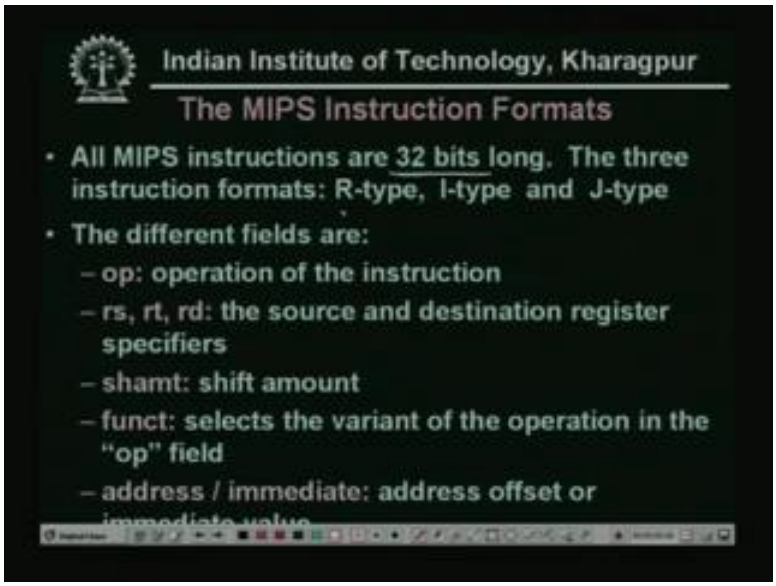
This class we will continue the discussion on the Design of Computer Instruction Set.

(Refer Slide Time: 00:58)

So, we have seen the how what is the format of a computer instruction, what are the two parts or two fields of a instruction and how the op code that instruction op code is encoded. And then, we introduce the, reduce set instruction in computer and what are the different RISC machine, so far developed. Now, already I mentioned that now a day's almost all the high performance computers are the RISC machines. So, just to see the what are the basic architecture of RISC, some early machines developed based on RISC we will see that thing.

So, some early RISC projects are IBM machine and Berkeley RISC 1 and RISC 2 machines and Stanford MIPS. So, this class we see the as an example the MIPS architecture.

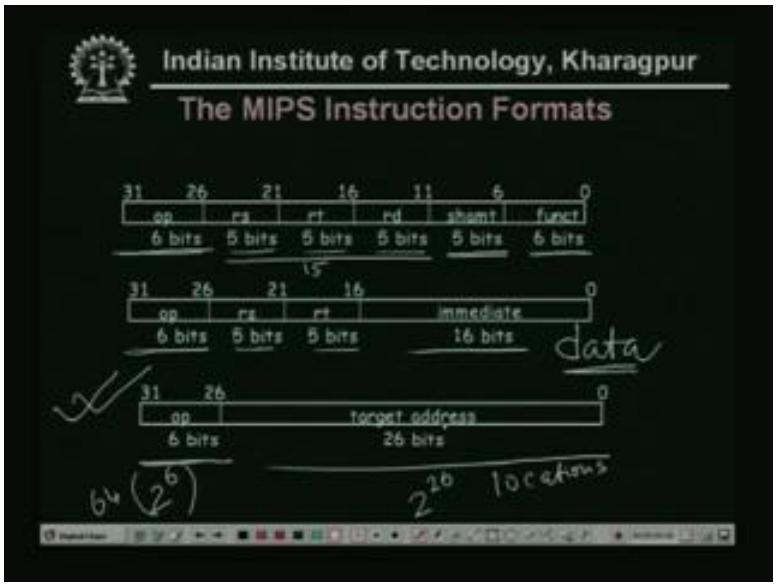
(Refer Slide Time: 02:25)



So, the MPIS instruction formats that all MPIS instructions are 32 bit long, so this is a instructions are 32 bits, the 3 instructions formats are R type, I type and J type. So, it supports three different instruction formats and the different fields are op means operation of the instructions. That means, which operation are supported or which operations are currently being executed by the instruction r s, r t, r d the three source and destination registers specifies.

Then, shift amount shamt is a shift amount that is a different field kept in MIPS, another is funct, it selects the variant of operation in the op field. Then, address and immediate; that means, address offset or immediate value; that means, the data or this is some kind of mode of operations, that whether it is immediate value that is being operated on or the address is given or address offset. And from some addresses the data to be accessed before it operate around, so these are the different fields that are kept on the early MIPS machine.

(Refer Slide Time: 04:28)



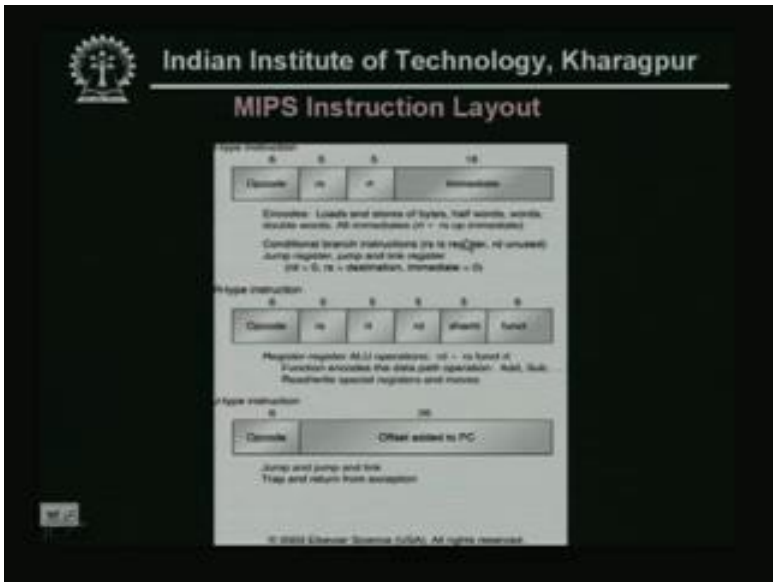
Now, these are the 3 type of the instruction format, so all are of 32 long, so see here this is a the first type that 6 bits are the op the operations which operations r s, r t, r d these are everyone is a 5 bits, means this 15 bits are for the registers, the source and destination registers. Then this is a shift amount this is again 5 bits and this is a funct, this is a 6 bit, now another one another instruction format is 6 bit is op code operation field.

Then, only r s r t 2 registers are kept instead of 3 and these are 5 bit, 5 bit rather immediate value; that means, 16 bits are kept for immediate. So, here that value is immediate value is can be taken, so that is why the 16 bits data, this is a actually the 16 bit data. Now, another one is that this is totally address field based, so what actually initially we introduced the instruction format is op code field and the address field.

So, the third one is the our the classical in nature, so 6 bits op code; that means, 2 to the power 6 or 64 such operations are specified and 26 bits address; that means, 2 to the power 26 locations are available to keep or to store the data to be operated. So, this is the thing, so this is our in the simple instruction format that has to field op code and address field, that this is the instruction the MIPS supports this one.

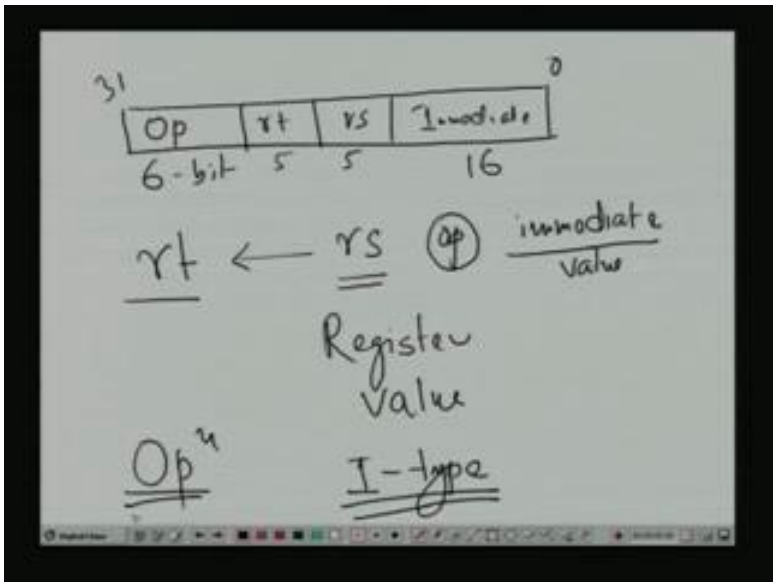
This is some a different type, that as if that values are stored on registers, the immediate values and that can be taken from this 2 to the power 16. The immediate values can be taken from here and in that way it can be worked out.

(Refer Slide Time: 07:21)



Now, the MIPS instruction layout these are of standard, so this is actually taken from the SBS Science USA because, this is already implemented. So, what layout they have fixed they design it is taken as it is it is given an example. So, this is a I type of instruction, as already we have mention they propose 3 of type of instructions format I type, R type and J type. So, I type means the immediate value type, see already we have seen that I type is op code r s, r t immediate, so it encodes loads and stores of bits half words W words and all immediate is r s to r t of immediate. That means, here the operations type is of it has this two type of r t op code r s, r t are the source and destination registers and the immediate type.

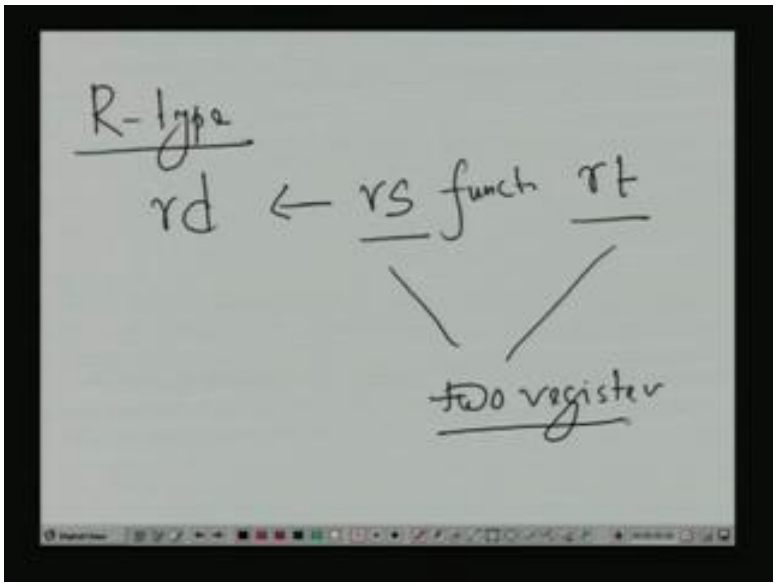
(Refer Slide Time: 09:20)



So, op code 6 bit 5 r t, r s 5 bit 5 bit and the rests are immediate value, this 16 bits because, total is 31 bits. Now, here the operations are of 2 registers it is a load store type, that r s the operations immediate, means whatever the immediate value is taken the operation is on the content of these registers. This is the resistor value, this is the immediate value and they two are operated and move to the destination registers. So, this is a type of operations the I type, so this is the I type operations to be performed it is a I type ((Refer Time: 10:51)).

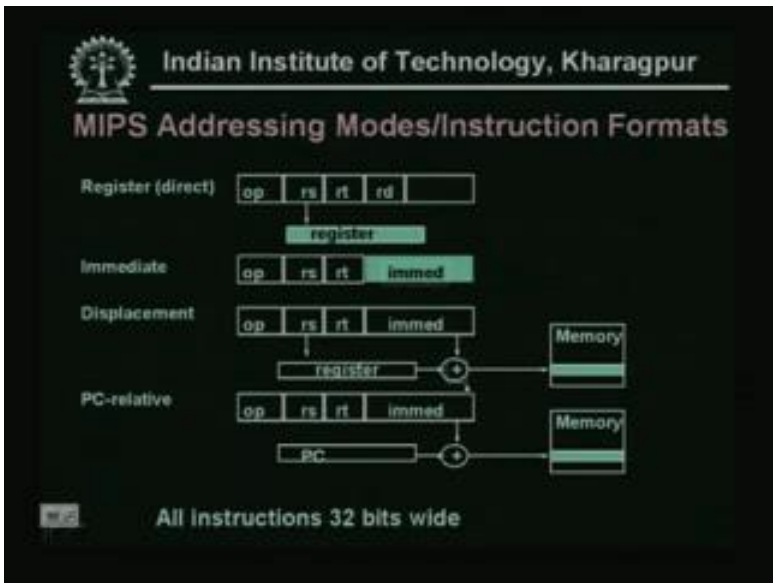
Now, conditional branch instruction r s is resistor r d unused and jump register jump and link resistor, R type instructions here that op code r s, r t, rd that shift amount shamt and funct, so resistor ALU operations is are the r s funct r t.

(Refer Slide Time: 11:37)



So, here the operations is the between 3 registers it is between 3 registers that this is a R type the operations are r d is r s funct r t. See, these are again two registers content of two registers functions ((Refer Time: 12:17)) and then it is operated on that it is value is move to r d, function encodes that data path operations add subtract and read write special registers and MOPs. And J type is a simple the op code and offset added to PC, so jump and jump and leak, trap and return from exception, so these are the three instruction formats initially proposed by MIPS.

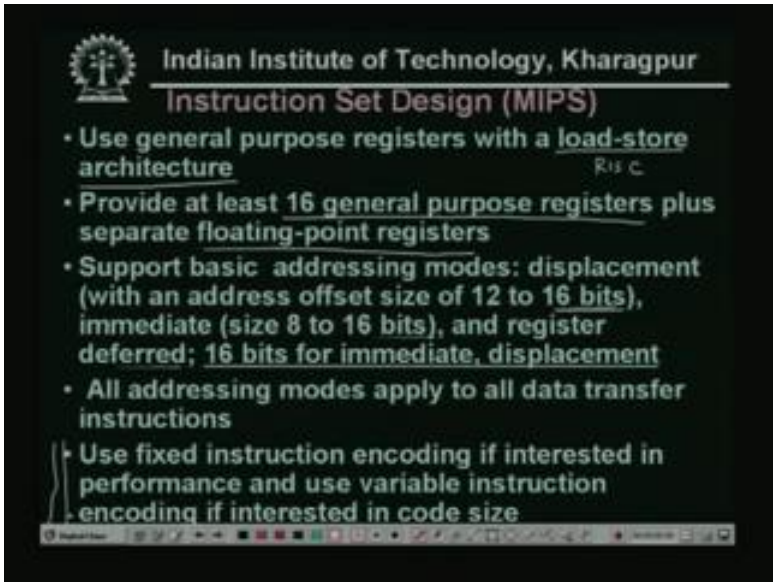
(Refer Slide Time: 12:48)



Now, MIPS addressing modes of instruction formats, so this is called a direct the resistor direct mode the 6 bit of op code is there and the 3 registers r s, r t, r d. So, r s is taken as if the resistor. Now immediate means, that already we have seen that always the data is the immediate value is operated, with the content of r t and shifted to r s, then some displacement. So, here see immediate value and the r s resistor r s r t are registers, so content of this registers, this is being operated and then that it gives the address of a memory.

So, these are immediate can specify the displacement, so this is a another facility or some flexibility has kept of this type that I type. Here, immediate is not the data, but the immediate gives as if the offset value, the address offset, Now PC relative that is the Program Counter and the immediate value and it always keeps a memory location and here all instructions are 32 byte.

(Refer Slide Time: 14:35)

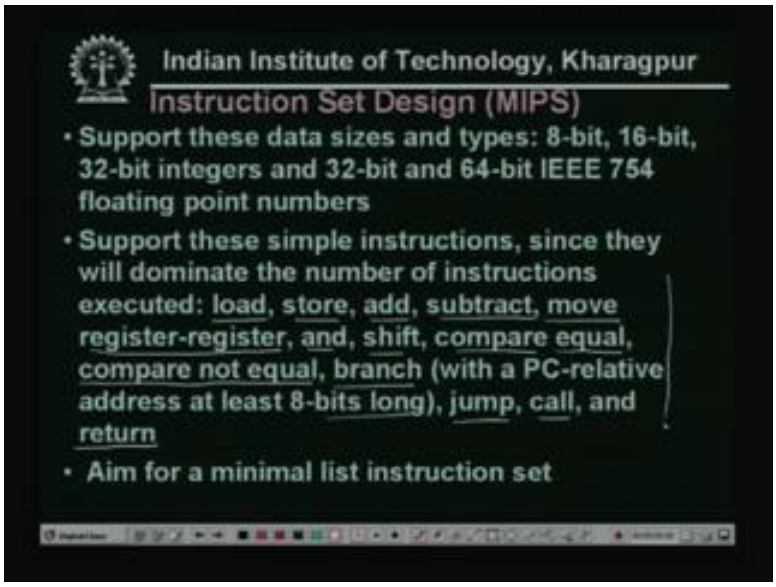


Now, how the instructions set are designed, so it use general purpose registers with a load store architecture, this is the basic principle of RISC architecture. So, this is the basic principle of RISC architecture is load store, provide at least 16 general purpose registers plus separate floating point registers. So, it keeps a number of registers, so 16 general purpose registers and also some floating point registers are kept.

It supports basic addressing modes like, displacement just now we have seen with an address offset size 12 to 16 bits and the displacement value is taken from the immediate

the 16 bit. Immediate size 8 to 16 bits and register deferred 16 bits from immediate displacement all addressing modes apply to all data transfer instructions. It uses fixed instruction encoding, if interested in performance and use variable instruction encoding if interested in code size. So, again it is a flexibility kept, normally RISC is told that fixed instruction encoding, but if needed it, it has some variable instruction encoding is also kept, if the code size is to be consider.

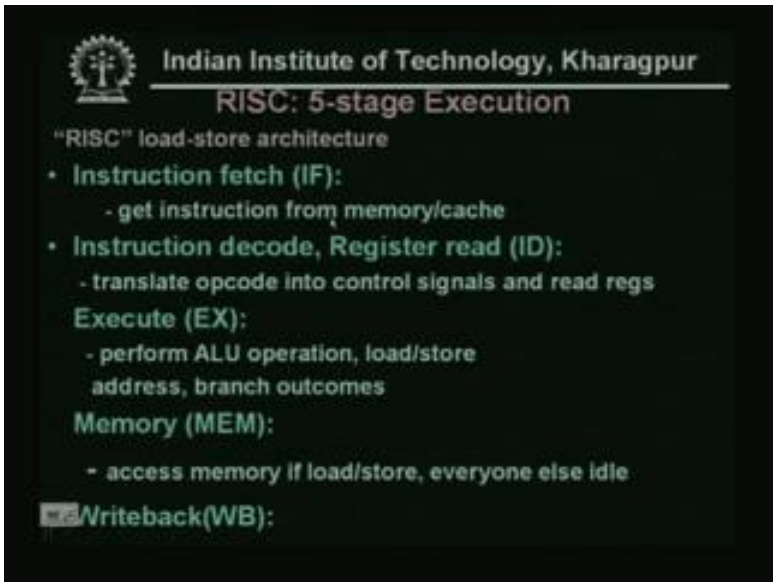
(Refer Slide Time: 16:22)



Then, it supports these data sizes and types 8 bits 16 bit 32 bit integers and 32 bit and 64 bit IEEE 754 floating point numbers, it supports the simple instructions. Since, they will dominate the number of instructions executed, the load, store, add, subtract, move, register to register and shift, compare, equal, compare not equal, branch with a PC relative address at least 8 bits long jump, call, and return.

So, it has some other miscellaneous instructions, but mainly the main instructions are load, store, add, subtract, and, shift, some compare some branching. That means jump type of instructions, then call and return aim for a minimal list instruction sets and from there they are selecting a basic instruction set and they are implementing this instruction sets. So, mainly the reduced instruction set the concept comes from here, the minimal list of instruction sets that are needed.

(Refer Slide Time: 17:53)

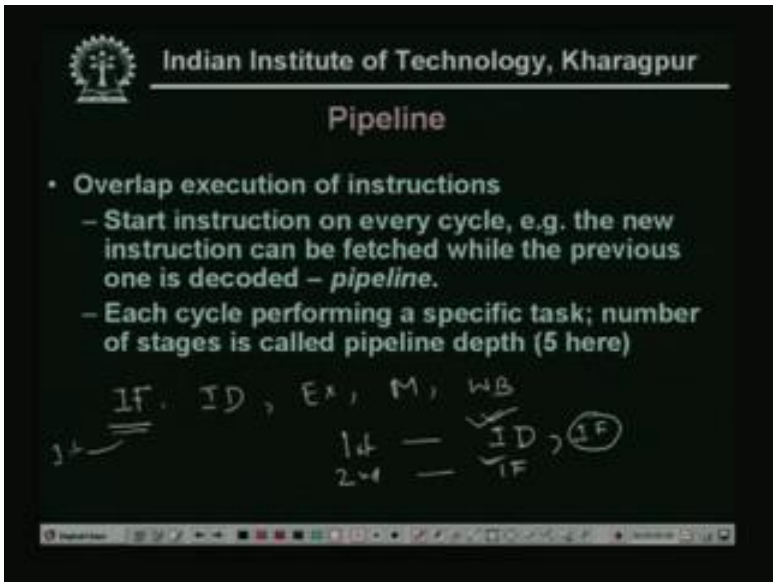


So, the approach is like that, now RISC is a 5 stage execution, this is already we have told that this is a load store architecture. Now, this five stage are instruction fetch, instruction decode, execute, memory and the write back, what it does instruction fetch it get instruction from a from the memory. That, which instructions to be are now is to be executed, so first it fetches the instruction, then instruction decode register read.

So, instruction decode that what is that instruction, translate op code in to control signals and read registers. As already we have seen one example, that every instructions or the op code field is actually encoded with some binary numbers. So decoding that binary numbers that which instruction actually it is means the instruction decoding, so we have done the op code encoding. Now, the reversing we will be doing that decoding the instructions, seeing the binary numbers assign to the op code field.

Now, execute this perform ALU operation and the load store addresses, branch outcomes. That means, whatever instructions is supported by MIPS, it will be executed by this stage in this stage execute. Memory, memory means this is a accessing memory, if load store and else everyone else idle and other is a write back.

(Refer Slide Time: 20:02)

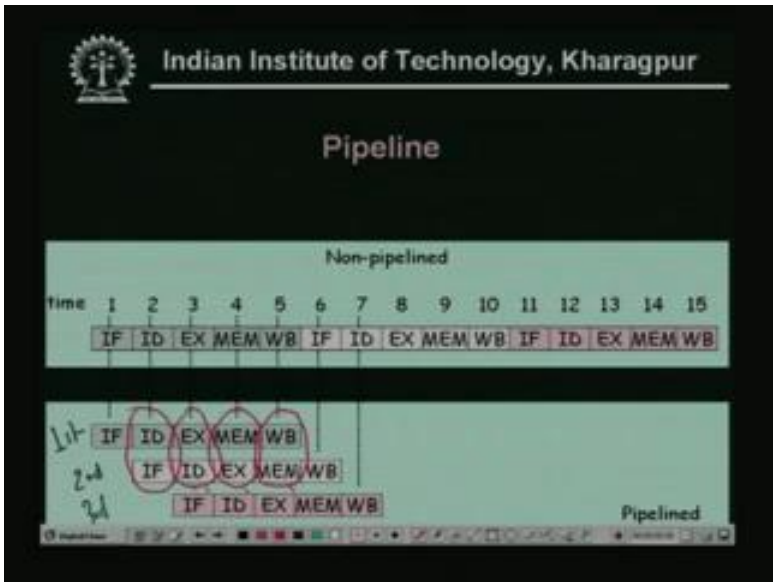


Now, another very unique concept and that is very important thing that pipeline concept now as come into picture. Now, concept is actually the overlapping or overlapping execution of instruction or we can tell the utilization of overlap execution of instructions is called the concept of pipeline, what is that? The start instruction on every cycle, the new instruction can be fetched while the previous one is decoded, so this is one stage of pipeline.

Each cycle performing as a specific task, number of stages is called pipeline depth, so already we have see it has some it has five stages, instruction fetch, instruction decode, execute, memory and write back. So, for one instruction say for one instruction we are doing we have already done instruction fetch and now for the second instruction say already it is fetched for first instruction. Then it should be decoded it has the first instruction is already, say for first instruction what we can tell, when it is instruction fetch has already done, it is over then it is instruction is decoded.

Now, the for the second instruction it can be a instruction fetch, so when the first one is decoded that time the second one is being fetched. So, this is some overlapping operation, so this overlap of execution of instructions that is called the pipelining and every cycle performing a specific task or number of stages is called a pipeline depth. So, here actually this is a five task are there 5 stages, so that is called the five here.

(Refer Slide Time: 22:42)



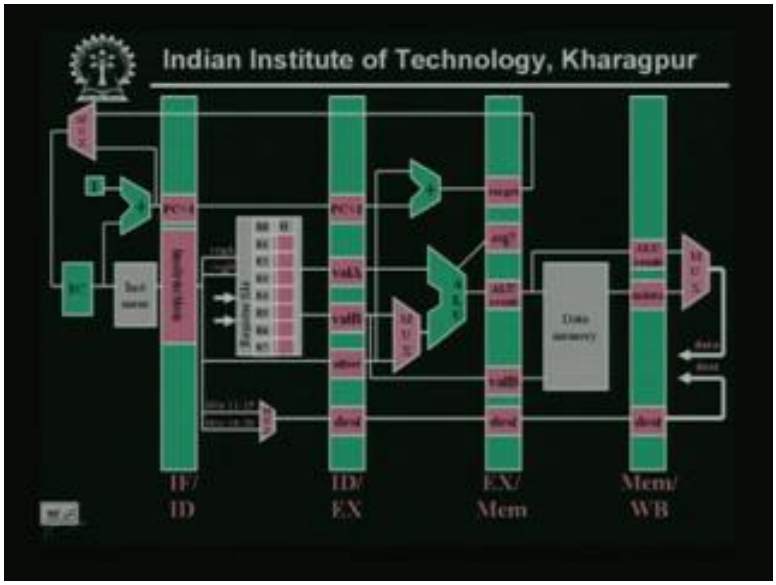
So, see this we have given here time scale the time access, as if this 1, 2, 3 up to 15 time steps we have shown. And this is the five stages instruction fetch, instruction decode, execute, memory and write back, similarly this is for another instruction the again that five stage, another five stage is kept. Now, see if it is a non pipeline; that means, for every instruction when the, say this is the first instructions then all the five stages should be executed and then only the second instruction can be in processed.

Again the third instruction cannot start the processing unless the second one is finished, so this is we can tell that the sequential execution of the instructions one by one. The one instruction one first, then instruction second, then instruction third like that, this is called the non pipeline. Now, the concept is as if the execution of one instruction that is being divide that is partition into five stages, that fetch decode, instruction fetch, instruction decode, execute, memory and write back.

Now, see that for the first instruction say this is for the first instruction, the instruction fetch is already it has been done. Now, the for the first instruction this is for the first the instruction is being decoded, now when it is being decoded then the second instruction is already in processing stage. So, for the second instruction the instruction is being fetched, now at the third step means third time unit, the first instruction is being executed and then the second one is in instruction decode stage and that time already third is also in process, the third instruction already is in process and now it is instruction fetch.

So, as if the first instruction is one time step in advance with respect to the second one, again similarly the second instruction set as if one time step in advance with respect to the third instruction set and so on. So, this is this overlapping say here the second time unit to the 5th time unit, the first instruction and second instructions are being overlapped. Similarly, from third time unit to the 6 time unit, the second instructions and the third instructions are in overlap and so on, if there are more such instructions include…So, this is the concept of the pipeline architecture and this is being employed or utilized in RISC.

(Refer Slide Time: 26:52)



So, if we see that the actual architecture of this pipeline, say these are here this is a instruction fetch or instruction decode, instruction decode and instruction execute, these execute and memory and this memory and write back. ((Refer Time: 27:28)) So, as see if we just see that thing say as if here at second time step these two are overlapped at, so ID instruction decode and instruction fetch. Similarly, a third time step instruction decode and instruction execute and as if these are come here, this is I D, I F this is I D, I F instruction decode instruction fetch.

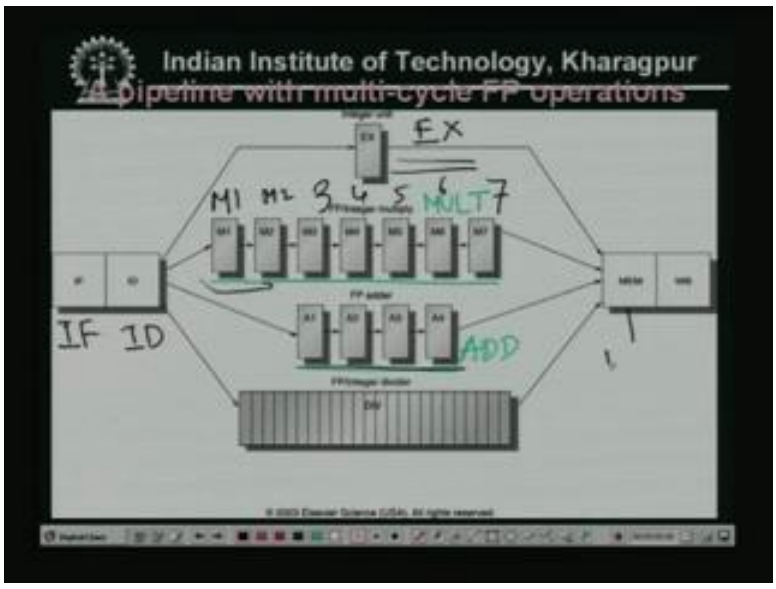
Similarly, at fourth time state memory and execute they are overlap and 5'th time step that write back and memory they are overlapped. And see this 4 overlapping is also here I D, I F, I D, I F, E X, I D execute I D, memory, execute write back memory, so actually there are 4 overlapping stage. Now, as if we are partitioning the time units and each time

units, which stages are being overlapped, so there are 4 overlap. And these overlapped are being shown this is I F, I D the overlapping of fetch and decode, overlapping and decode and execute, overlapping of execute a memory, overlapping of memory and write back.

So, see this is the program counter some multiplexer is there, this is instruction memory, so from this instruction memory, instruction is being fetched. And as a instruction a program counter is implemented by one, then it goes to some these are some register files. So, it goes to the after instruction fetch, it goes to the decoding place say instruction decode, now again this is after decoding that what type of operation it is. So,, it will be execute that type that say memory is also from data memory, that data is being taken from the memory.

And then it is a write back; that means, again the result is stored on the memory or the data being updated on the memory, so this is one stage. Now, when one instruction as advanced one time step; that means, a instruction decode, then from the instruction memory, so one this is a first instruction and now it has come here. Now, the second instruction is being fetched, and then it is being, so when it is being decoded then it is second one is fetched, so this is the overlapping region and in this way it will proceed for other instructions also.

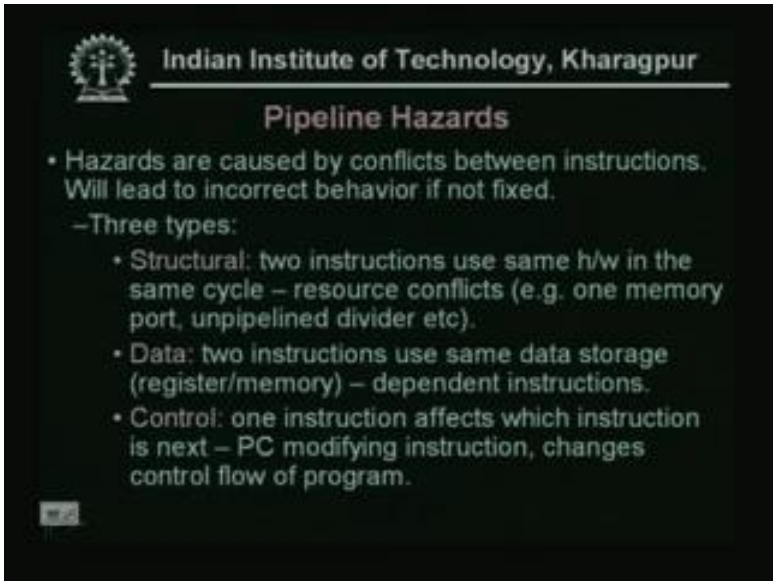
(Refer Slide Time: 31:12)



Now, a pipeline with multi cycle floating point operations, say we are the only the two stage we are considering. Say this is first we are this is a instruction fetch and say instruction decode, now say two operations are shown here, one is multiply a MULT and here this is a ADD addition. So, this is our execution mode EX, this is the execution phase, so the first instruction is fetched decode and it is being executed, say it has a multi cycle operations, say for multiply we need m 1, m 2, 3, 4, 5, 6 say 7 cycles.

Similarly, for addition say there are 4 such cycles A 1, A 2, A 3, A 4, so when one instruction will be in process, that mean it will be advance with one cycle. Then the next instructions can also be processed for this floating point operations, so here actually this cycles or this can be a pipelining stage, for this floating point operations, this is a multi cycle floating point operations. Now, here the data memory is accessed, this is a memory operation and then it is a write back, so this is one example of a pipeline with multi cycle floating point operations.

(Refer Slide Time: 33:49)

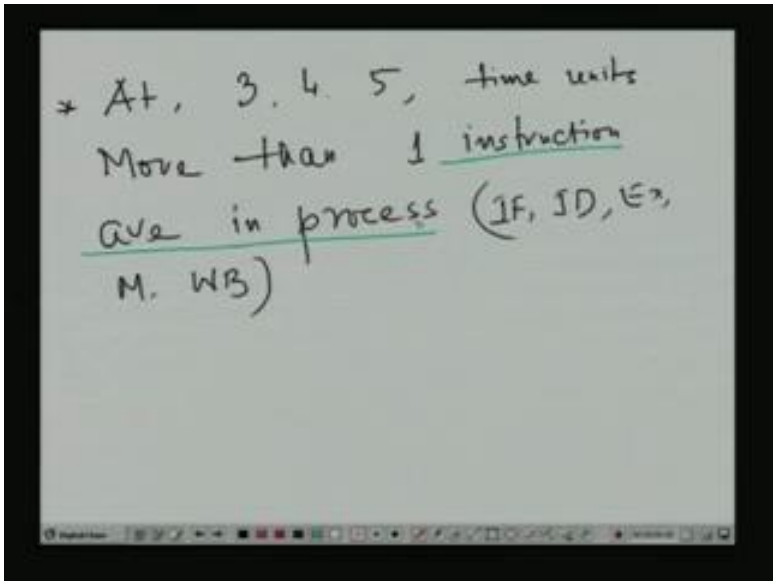


Now, there are some problems associated with pipelines, normally we call that are hazards, hazards are mainly these are some problems to be tackled, when we employ when we use the pipeline architecture. So, hazards are caused by conflicts between instructions, which will lead to incorrect behavior if not fixed, so these are some problems to be handle. And as pipeline is a game, that we are actually reemploying the

overlapping stage; that means when one particular instruction is in one stage, say it is instruction decoding, then the second one is started the it is fetching part.

So, that is the advantage we are doing, that there are some overlapping stage, now for that there can be conflict because, two instructions are processed parallel, at least two there are some stages, where the instructions are say we have seen that here, there are…So this is a overlapping stage ((Refer Time: 35:33)) where 3 stages are actually overlapped, first instruction is in execution, second instruction decode stage, whereas the third instruction is in instruction fetch. Similarly, at the four time step also, there are overlapping of three stage, so at the 3, 4, 5, time steps, these are the say this is a 3 stage overlapping of three stage.

(Refer Slide Time: 36:37)



So, these time steps; that means, are 3, 4, 5 time steps or time units more than one instruction is in process, either fetch or decode or execute or memory and write back different type of stages. So, what will happen that as more instructions are in process, this is the main point, so there instructions can access the same data or the same register value, then what will happen if parallely that one than one instructions, access the same thing.

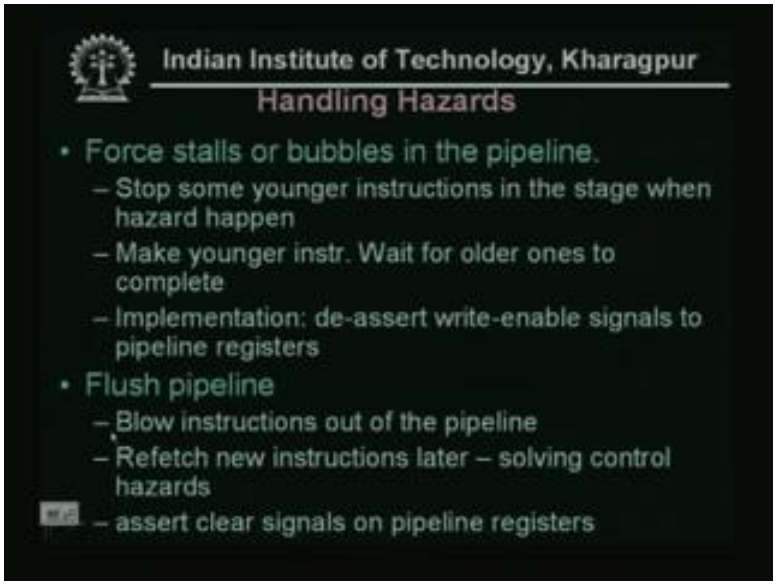
So, this main problem or the main hazards means that this is the problem, if more than one instruction want to do the same thing. ((Refer Time: 38:04)) So, hazards are caused by conflicts between instructions, it will lead to incorrect behavior if not fixed, so

normally there are three types of hazards, structural hazards, data hazard and the control hazard. Now, what do you mean by structural hazards, the two instructions use same hardware in the same cycle, this is called a resource conflicts, say one memory port or un pipelined divider, etcetera.

Say, one operation you need say one multiply or one divider is there, now two instructions one to use or one to employ the same unit or say one memory cycle same cycle. That means, the same hardware unit being requested or being used by the two instructions parallely or the same time, same cycle, then it is called a structural hazard. Now, data hazard now two instructions use same storage, register or memory, the data is stored either in the register or in memory and two different instructions because, they are in process and that is the concept of pipeline.

So, the operation has such that both the instructions use the same data storage, then it is a data hazards, now the control hazard that one instruction effects, which instruction is next. So, PC modifying instruction changes control flow of program, so this is also a one type of hazard called the control hazards.

(Refer Slide Time: 40:29)



Now, how to handle the hazards, so there are mainly two things to be implemented one is force stalls or bubbles in the pipeline. So, if the first intuitively what a things, that if more than two instruction want to get the same resource, hardware or data from the memory. Then, actually hazard comes then what, we can do we stall one; that means, one

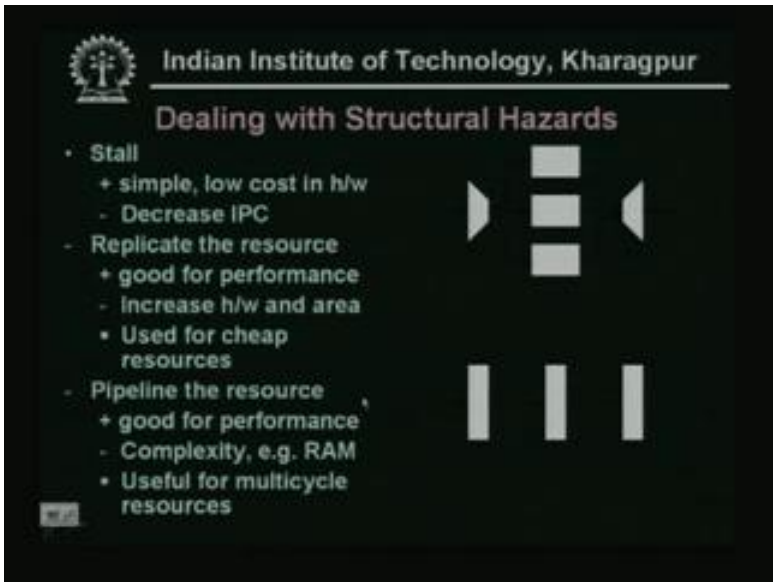
will use that thing and one will stop it is using, then hazard can be control, so this force stalls or bubbles in the pipeline.

So, stop some younger instructions in the stage, when hazard happen, that means as if the anyone say we are calling younger one. Next one which comes a later, that instructions access a is being stopped, whether it is a hardware or whether or it is a data resource from memory or register. Make younger instruction, wait for older ones to complete, again that stalling means that actually it will wait for some predefine time or it will wait for some time, until the first one completes is use.

Then, in that way we can handle the hazard and de assert write enable signals to pipeline registers, this is called the implementation. That means, how it is being implemented the handling hazards, then it is some write enable signals to pipeline registers in that way it can be implemented, another is flush pipeline. So, blow instructions out of the pipeline; that means pipelines means some overlapping of some stages.

So, if this hazard occurs; that means, just flush out as if one stage means one temporary that time units at that time unit some of the instructions are being thrown out, re fetch new instructions later solving control hazards. So, these instructions are again re fetched and assert clear signals on pipeline registers. So, this is how it is being implemented that clear signals on pipeline registers are being implemented, just to implement the flush pipeline concept. Just like the force stalls that is the, as the write enable signals to the pipeline registers are implemented to get the force stalls bubbles in the pipeline.

(Refer Slide Time: 44:03)



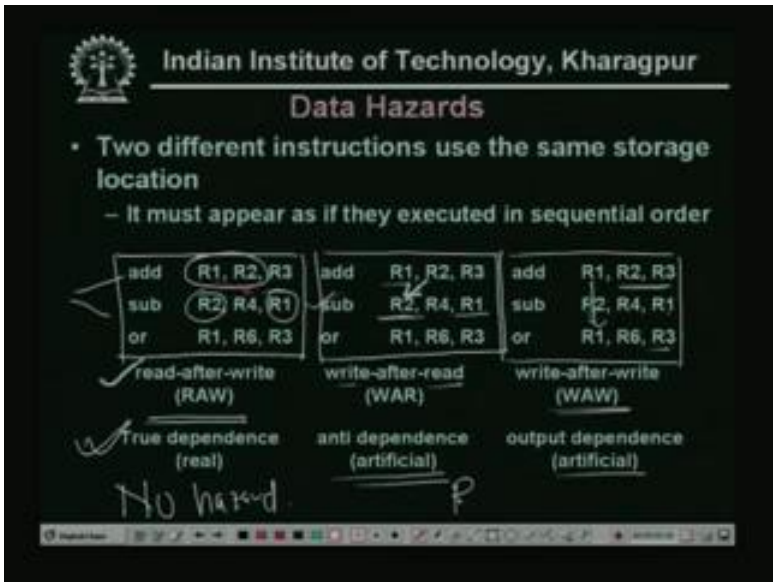
Now, we see how the structural hazards are dealt, see that it is a stall if it is a stall type, so it is simple low cost in hardware and decrease the IPC, Replicate the resource. So, good for performance, increase hardware and area used for cheap resources, so replicate the resource because, two instructions parallelly want to use the same resource, but we have only one resource. So, one solution can be that replicate the resource, so; obviously, it will be very fast because, there will be no hazard is control.

And the performance will also be good because, parallelly or at the same time both are using the resource, but replication means that it will increase the hardware. And; obviously, hardware means it will be the increase the area, but speed wise it will be very good, used for cheap resources. So, it can be a good solution if the hazard occurs for some cheap resources, say that now a day's hardware say memory, this is a very cheap. So, if the two for some instruction they want to use the same memory, then what we can do that we can replicate this one this, so this will be good solution.

Now, pipeline the resource, this is good for performance, complexity and again for restricted RAM that only that RAM is there. So, it will be a good use, useful for multi cycle resources, just now we have seen one multi cycle floating point adder and the multiply. So, for that type of structure for that type of machines, that if two instructions want to use the same floating point unit and the hazard occurs, then as it is a multi cycle then always what we can do, we even that cycles that can be pipelined.

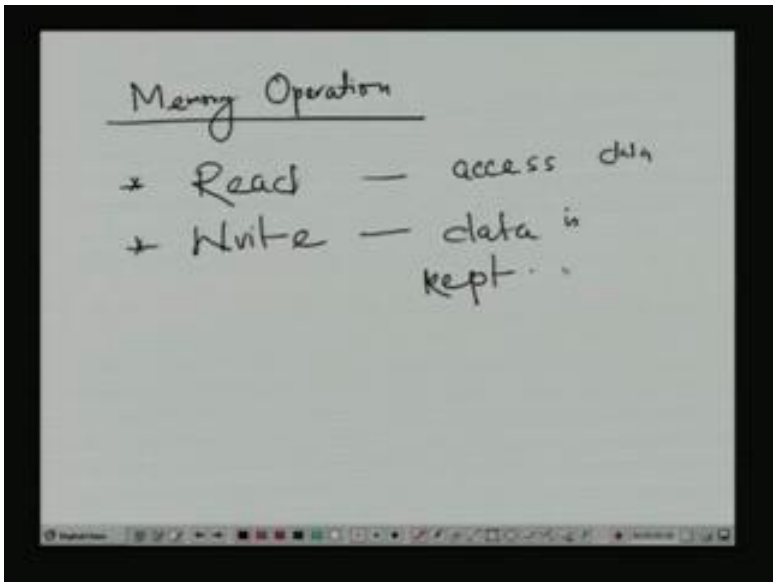
So, that for not only that overall pipeline concept, that is a this is a nested type of thing, that for floating point units that whether it is a floating point multiplied, floating point adder, as if the cycles are also pipelined between the resources, pipeline the resource. As if these are the resource and it is pipelined between the two instructions, so this can be the solution. So, here actually we have shown that if this type of hardware is there, that everyone is using that and then this can be easily can be pipeline.

(Refer Slide Time: 47:51)



Now, the data hazards, so two different instructions use the same storage locations, data hazards means this is that either same storage locations, either from memory or the registers. And there are different cases we see, it must appear as if they executed in sequential order, so two different instruction use the same storage locations.

(Refer Slide Time: 48:31)



See one memory operation means always the memory operations are two only two already we have read the memory the ROM the RAM. And we have seen actually there are two operations read and write; that means only the memory is access always it is accessing the cell the data or the data is kept or stored ((Refer Time: 49:23)). So, see this is what we are seeing this is one type of see this is a add R 1, R 2, R 3, so the register values are added.

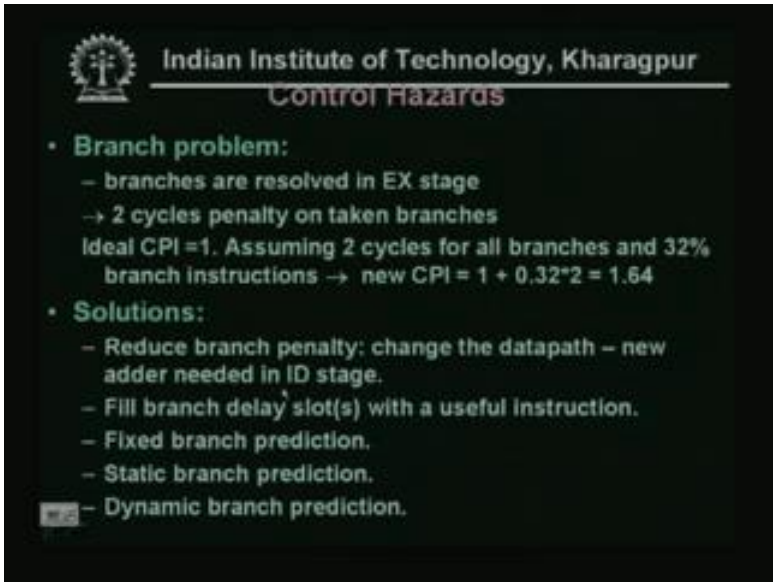
The next instruction is subtract R 2, R 4, R 1, see here that both the content of see the R 1, R 2 are common, here R 1 and R 2 are common. So, and they are subtracted and then kept or R 1, R 6, R 3, see here it is a R 1 add, so first the two values are read, the values are added and then it is kept. So, here it is a write is also there, now between these addition and subtraction actually it is a read subtraction it the value will be read after this write. That means, in subtract this instruction there will be a write after a read operation in the add instruction.

So, here the two instructions add and sub they use the same storage R 1 and R 2, now is there any conflict because, both are using the same storage R 1, R 2 the two registers, but here subtract it is a read, which was written the first instruction add. So, this is a read after write, we can tell this is a read after write and this is a two dependence or real, so there will be no problem no hazard here, hazard is control.

Now, next one say add R 1, R 2, R 3 and say this here R 1 this R 1 value was here, now R 2 is subtract R 2, R 4, R 1. So that means, here this is a write after read because, here also the R 1, R 2 are common registers, R 1 R 2 are common registers and in the first instruction add, the value reads up to and that is being updated in the second instruction. So, this is a write after read and this is a anti dependence, so this is a artificial.

Similarly, the third one say here add R 1, R 2, R 3 subtract R 2, R 4, R 1 or R 1, R 6, R 3, so here R 2, R 3 value that added and kept in R 1 and again or R 6, R 3 is odd kept in R 1. So, R13 is write after write, that is in the first instruction and the third instruction add and or, so again this is a write after write, this is also output dependence and this is artificial. Another one is there that is called read after read, so read after read as normally this is not a problem because, unless we write, sometimes, that will be of no… it will not create any hazard, so these are the part of data hazards.

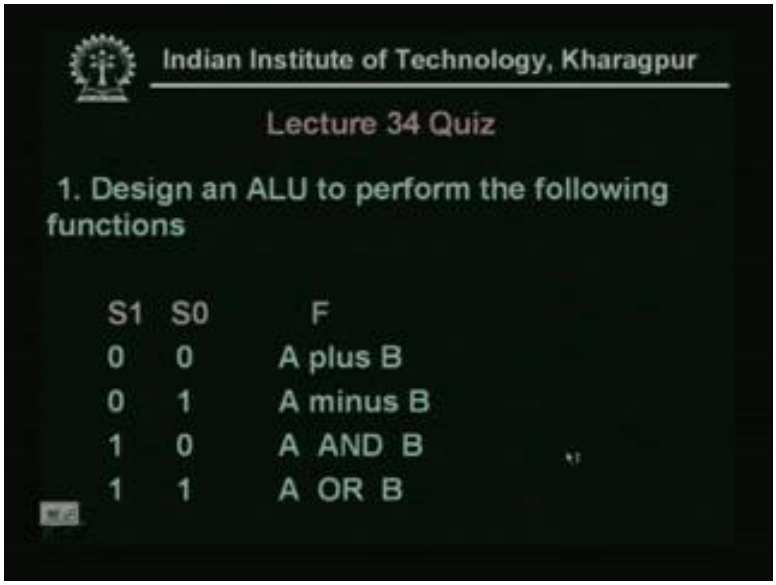
(Refer Slide Time: 54:56)



Now, the control hazards, so this is a branch problem and branches are resolved in E X stage. So, 2 cycles penalty, so this is a branch problems and solutions are reduced branch penalty, so change the data path, new adder needed in a I D stage, the instruction decode path, field branch delay with a useful instruction or fixed branch prediction, a static branch prediction or dynamic branch prediction. So, using these solution control hazards can be handled.

So, mainly if a in this class what we have, we have a read the reduced instruction set architecture, we have see that for mainly one pipeline operation can be implemented that is a very big advantage. But, for pipeline there can be some problems called the hazards the structural data and control can be there and how this hazards can be control that we have seen.

(Refer Slide Time: 56:02)



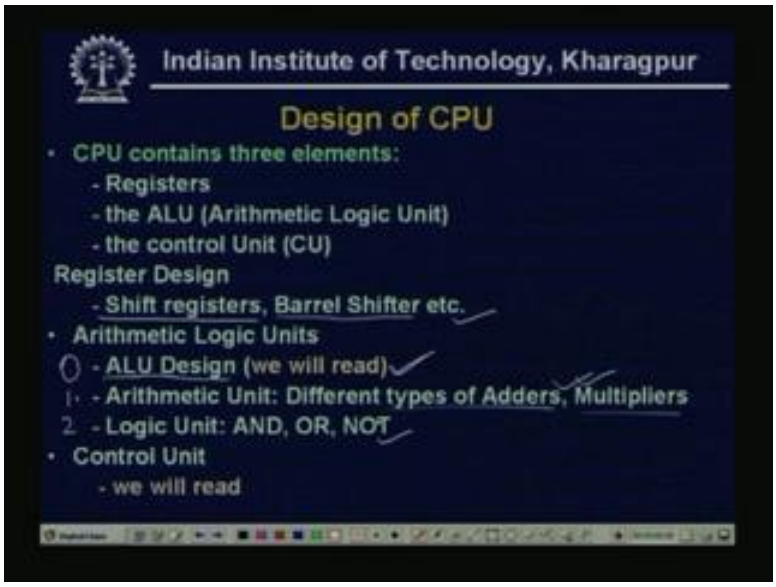
So, now the lecture 34 quiz giving one problem already we have seen that how it can be designed, actually in the next class we will see that how the arithmetic logic unit will perform the functions. So, actually in the next class we will be seeing that how the ALU can be designed, this is a part of CPU design, so we will end the class here.

Digital Systems Design
Prof. D.Roychoudhury
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 35
Design of Computer Instruction Set and the CPU (Contd.)

We are reading the Design of Computer Instruction Set and the CPU. In the last class we have seen that how the computer instructions set can be designed and the reduced instruction set machines, then the current state of the art actually the all the machines now a days are reduced instruction set. So, how are the instruction set are selected, how it can be designed? How is the hardware implementation of it? That already we have read.

(Refer Slide Time: 57:33)



Today, we will start discussion on the design of CPU and first we see the design of ALU in this class. So, CPU contains a three elements, the registers, the ALU the Arithmetic Logic Unit and the Control Unit or CU, now already when we have discussed with read the different type of register designs, there we have seen the how shift registers can be designed using flip flops. The barrel shifter or what we can tell that a this is actually a more than one bit transfer that called the barrel shifter, the 4 by 4 or 8 by 8 barrel shifter 16 by 16 how they can be design already we have read that part.

Now, the arithmetic logic units, as it name implies the arithmetic units, so it has actually two parts, one is the ALU design, the arithmetic unit and then logic unit, now the.