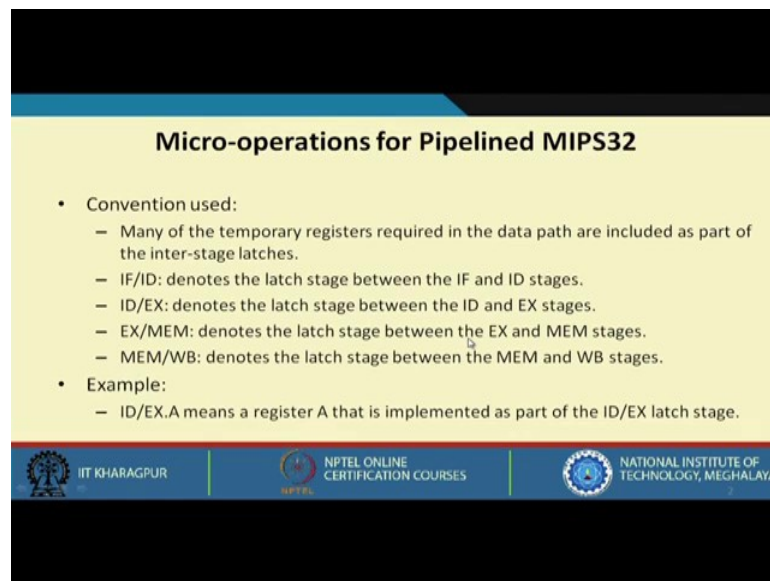


Computer Architecture and Organization
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 54
MIPS32 Pipeline (Contd.)




In our last lecture we had identified some of the requirements that need to be satisfied in order to have a pipeline version of our MIPS32 architecture. Now we shall be looking into that, how we can modify the non-pipelined data path of the MIPS32 that we have seen earlier into an equivalent pipeline version that works. Of course, subsequently we shall try to identify what are the problems or conflicts that can come in? And how we can solve them?

(Refer Slide Time: 01:10)



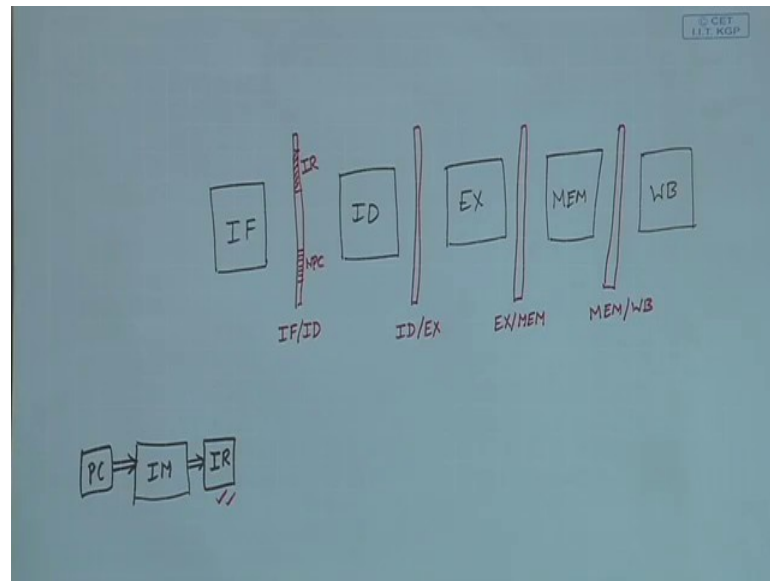
Micro-operations for Pipelined MIPS32

- Convention used:
 - Many of the temporary registers required in the data path are included as part of the inter-stage latches.
 - IF/ID: denotes the latch stage between the IF and ID stages.
 - ID/EX: denotes the latch stage between the ID and EX stages.
 - EX/MEM: denotes the latch stage between the EX and MEM stages.
 - MEM/WB: denotes the latch stage between the MEM and WB stages.
- Example:
 - ID/EX.A means a register A that is implemented as part of the ID/EX latch stage.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES |  NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

We shall be using some conventions in our discussion.

(Refer Slide Time: 01:34)

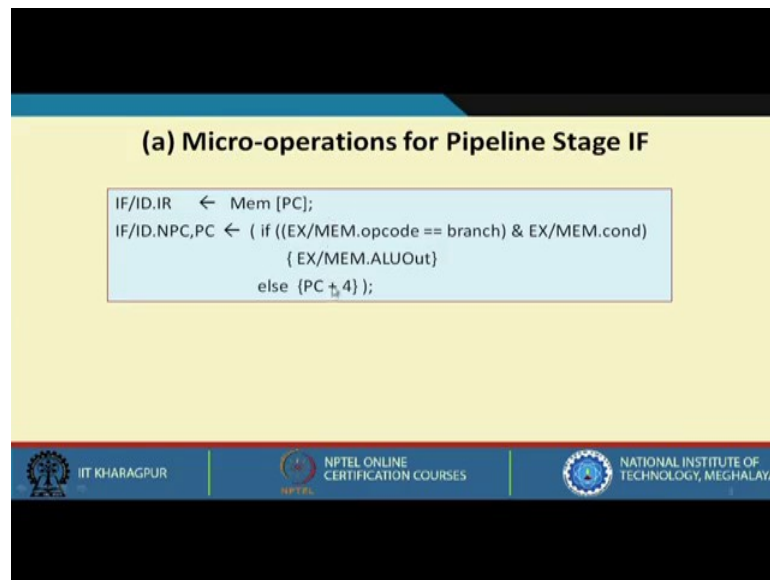


You see there are five stages: IF, ID MEM, EX and WB. You also require some latch stages in between, some register stages. The convention that I am talking about is, each of these latch stages we are giving some specific names. For example, this latch stage between IF and ID, we call them as IF/ID. This latch stage, we call as ID/EX, this one as EX/MEM, and this one as MEM/WB. In the non-pipelined version if you recall, in the IF stage, what we are doing? There is an instruction memory, that is being accessed by some address from PC; and some data is read, it is loaded into a register called IR; instruction register.

But here we need not use separate registers like IR for example, in every stage. Rather in this IF/ID stage itself we can mark a portion of it. Say, these are register type, these consists of certain number of bits. We can mark certain number of bits here that can be our IR. We can here mark certain number of bits here that can be our NPC. The idea is that we shall not be using these registers separately within these stages; rather many of these registers we shall be integrating along with these latches.

Let us come back to the conventions. Already I have mentioned IF/ID, ID/EX. These are the names of the latches. So, if I write ID/EX.A, this means that I am referring to a register A that is implemented as part of the ID/EX latch. This is the naming convention that shall be followed.

(Refer Slide Time: 04:46)



(a) Micro-operations for Pipeline Stage IF

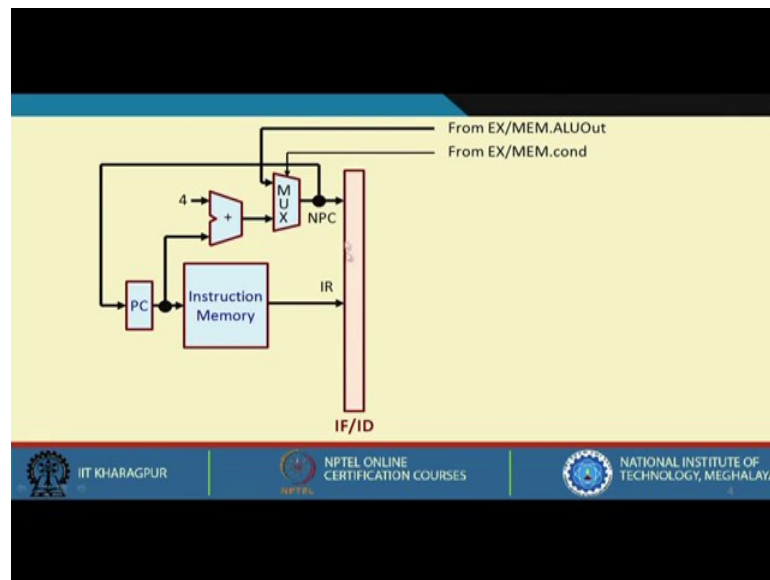
```
IF/ID.IR ← Mem [PC];  
IF/ID.NPC,PC ← ( if ((EX/MEM.opcode == branch) & EX/MEM.cond)  
                { EX/MEM.ALUOut  
                else {PC + 4} );
```

The slide features a yellow background with a blue header and footer. The footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA.

Let us look into the stages one by one. In the IF stage, one thing we mentioned that we need to increment our PC by 4 here itself; we cannot wait till MEM. So, our micro operation can be like this. We are doing an instruction fetch, after fetching where we are storing? We are storing in to IR, meaning IF/ID.IR. IR is part of the IF/ID latch. Similarly in the IF/ID.NPC, there is a register NPC as part of this, while also there is a separate register PC. This is loaded with either this ALUOut from EX/MEM or PC + 4, depending on some condition. If it is a branch instruction and the branch condition is true, then you will have to load PC with EX/MEM.ALUOut. If it is a branch instruction you have no way to know the branch target address in IF; you will have to wait, but if it is a not branch instruction you can proceed. You can proceed with PC = PC + 4, and we are ready with the next stage, when the next instruction can be fetched.

So, exactly we are doing that. We are making a check if it is a branch instruction and condition. You see this checking is not that difficult because we are already fetching it here, and this opcode equal to branch means we are checking a few bits in IR whether it is a branch condition. Of course, we are checking EX/MEM.opcode, so later. By default in IF we will be going to the else part PC + 4, but if it is a branch instruction and if the condition is a taken branch, then something else will be loaded. The IF stage will look like this.

(Refer Slide Time: 07:31)




This is our IF and this is the IF/ID latch. We have a register here. This is IF/ID.IR. Similarly NPC is another register inside IF/ID.NPC, and PC is a separate register of course. PC is used to access memory and this multiplexer is selecting either PC + 4 or this ALUOut that is coming from EX/MEM. By default PC + 4 will be loaded. This is how we are making the modification in the IF stage so that PC = PC + 4 is done here itself, but if it is a branch instruction some corrections have to be made.

(Refer Slide Time: 08:35)

(b) Micro-operations for Pipeline Stage ID

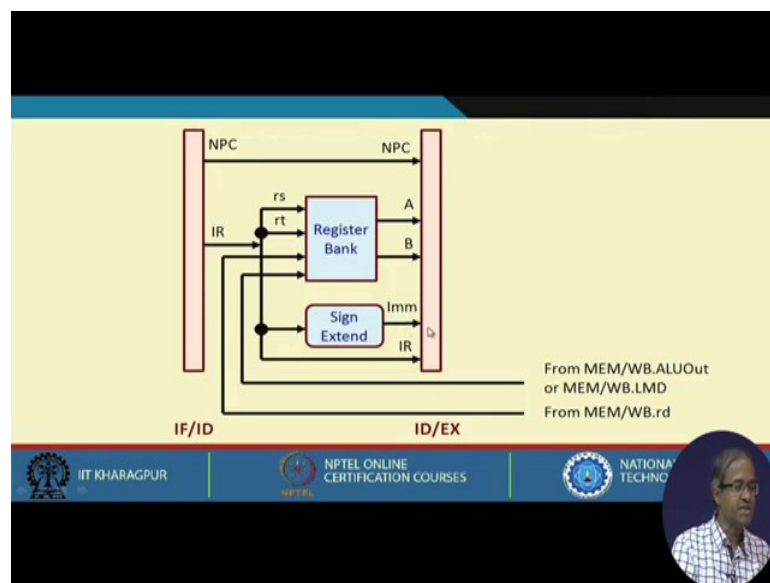
```
ID/EX.A ← Reg [IF/ID.IR [rs]];
ID/EX.B ← Reg [IF/ID.IR [rt]];
ID/EX.NPC ← IF/ID.NPC;
ID/EX.IR ← IF/ID.IR;
ID/EX.Imm ← sign-extend (IF/ID.IR15..0);
```



The slide shows the micro-operations for the ID stage. It lists five operations: ID/EX.A, ID/EX.B, ID/EX.NPC, ID/EX.IR, and ID/EX.Imm. Each operation is defined by a register file access or a sign-extension operation. The slide also features logos for IIT Kharagpur, NPTEL Online Certification Courses, and National Institute of Technology, Meghalaya, along with a small portrait of a man in the bottom right corner.

Now, for the ID stage what were the operations? We are doing some register prefetch. In the previous one we saw this IR is in IF/ID. From there we are checking the different bits; source registers and the other thing, Imm. That is why IF/ID.IR; the rs field of that IF/ID.IR, the rt field of that. We are prefetching the two registers and storing them in A and B registers, which are part of IDEX latch. Next stage latches and NPC you are not doing anything simply, we are forwarding this NPC from IF/ID to ID/EX because this will be required later. Similarly for IR we are doing the same thing. For Imm, from IF/ID.IR, we are doing sign extension and we storing in ID/EX.Imm.

(Refer Slide Time: 09:48)

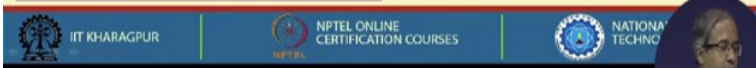



Whatever I have written here can be pictorially shown like this. You see in the previous IF/ID stage, I have this NPC and IR, but in the next one ID/EX, I have NPC, A, B, Imm, IR. IR I am simply copying, NPC I am simply copying, A and B I am reading from the register bank, this Imm I am doing sign extension. You see this is exactly what you have done. Fetch, sign extension, these are simple copies, and these two arrows are actually for the registers. This will come from later stages. Here only we shown register read but because the register bank is located in stage ID, the register write signal should also come here, from MEM/WB.rd and MEM/WB.LMD or MEM/WB.ALUOut.

(Refer Slide Time: 10:54)

(c) Micro-operations for Pipeline Stage EX

$EX/MEM.IR \leftarrow ID/EX.IR;$ $EX/MEM.ALUOut \leftarrow ID/EX.A \text{ func } ID/EX.B;$ R-R ALU	$EX/MEM.ALUOut \leftarrow ID/EX.NPC +$ $(ID.EX.Imm \ll 2);$ $EX/MEM.cond \leftarrow (ID/EX.A == 0);$ BRANCH
$EX/MEM.IR \leftarrow ID/EX.IR;$ $EX/MEM.ALUOut \leftarrow ID/EX.A \text{ func } ID/EX.Imm;$ R-M ALU	
$EX/MEM.IR \leftarrow ID/EX.IR;$ $EX/MEM.ALUOut \leftarrow ID/EX.A + ID/EX.B;$ $EX/MEM.B \leftarrow ID/EX.B;$	LOAD / STORE

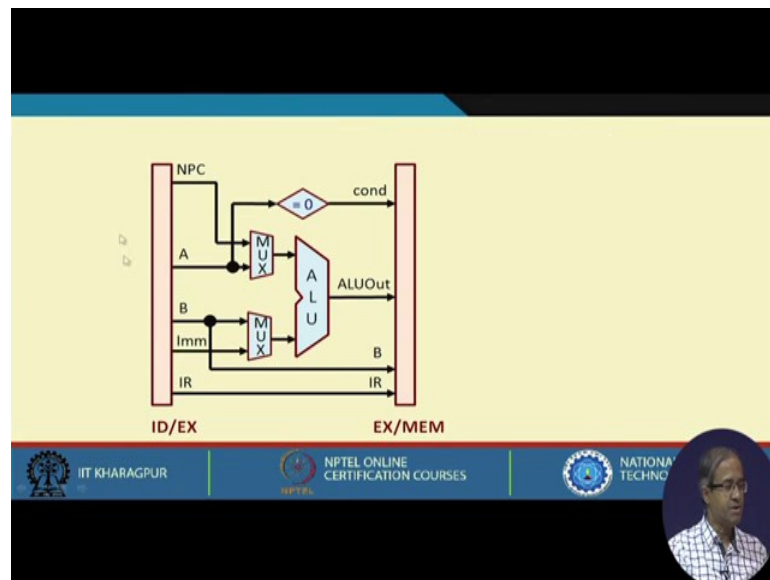




Now, let us come to the EX stage. Here the operation is different depending on the kind of instruction. If it is R-R ALU, then we are doing some operation on A and B, and storing it into ALUOut which is part of EX/MEM. And IR again we are forwarding to the next latch.

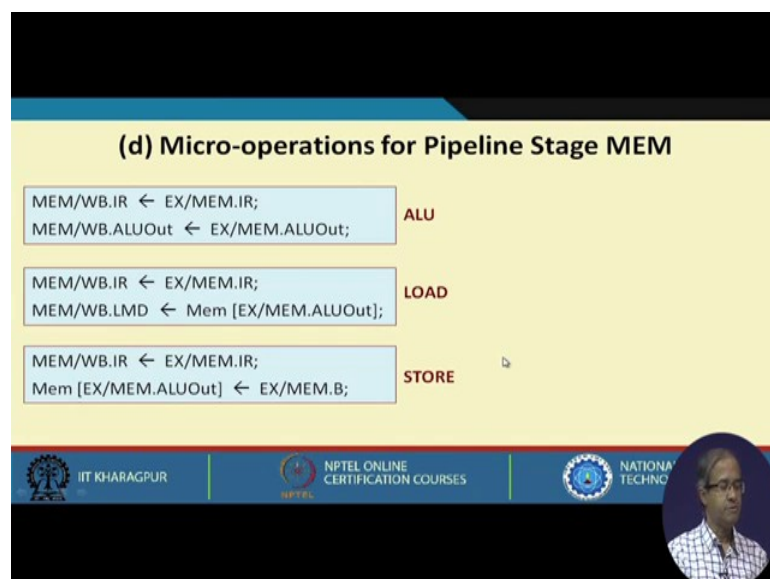
For R-M ALU, one operand is register and other is immediate data. We are actually adding a register content or subtracting some operation depending on the function to an immediate data, and for LOAD and STORE you will have to calculate the effective address. You are forwarding it, this ALUOut you are calculating, and for branch, you are doing that same thing; calculating the branch address and checking the conditions.

(Refer Slide Time: 12:31)



In terms of the pipeline implementation, this EX stage looks like this. Here the operation may look quite complex, but in terms hardware it is very simple. In the ID/EX stage you had NPC, A B, Imm, IR. This IR gets copied and B gets copied, because you need B for some instruction. But ALUOut gets stored and cond gets stored. This IR and B are getting forwarded. This is my EX stage, just an ALU, some multiplexer and a zero condition checker.

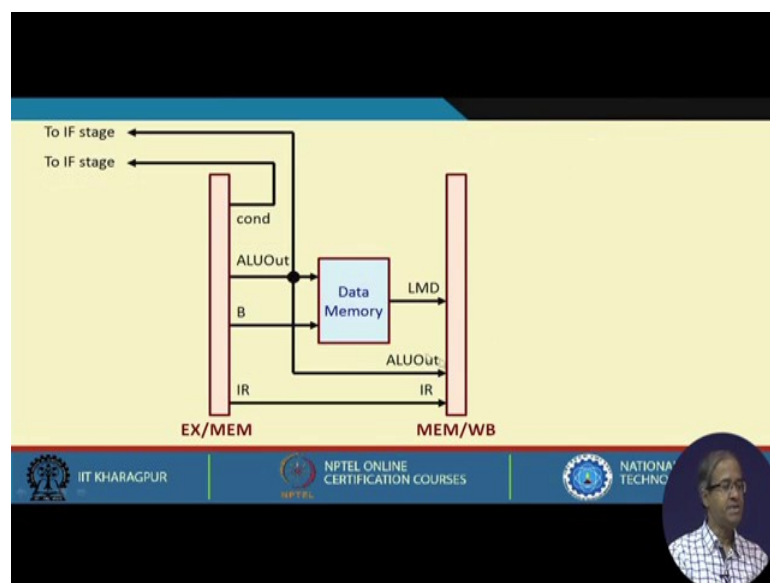
(Refer Slide Time: 13:19)



When you come to the MEM stage, for ALU operation you again simply forward IR to the next latch and also you forward ALUOut here, because for ALU instructions MEM does not do anything. It simply forwards it, for load operation you are doing a memory read and IR is forwarded, for store operation you are doing a memory write, and IR is forwarded.

Here memory address is in EX/MEM.ALUOut. While reading you are storing it in LMD, and while writing you are taking it from EX/MEM.B.

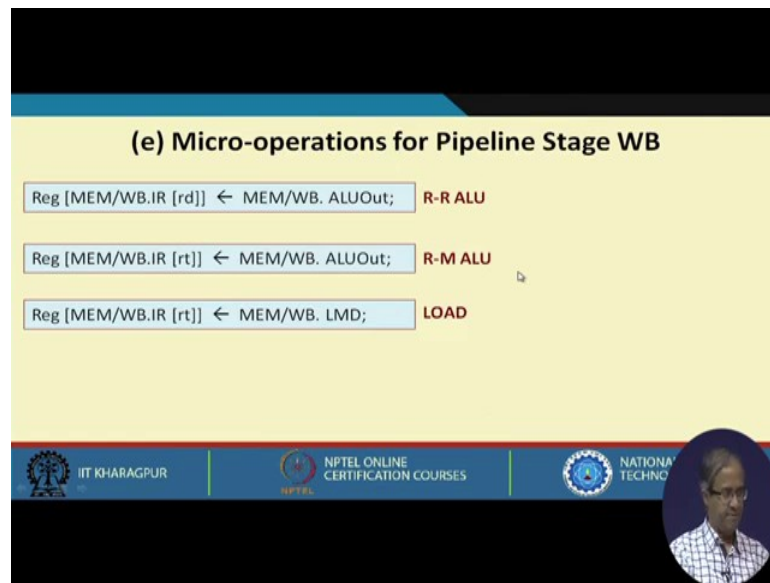
(Refer Slide Time: 14:04)



The MEM stage is simply like this; it consists of a memory. Your address is coming from here, data is coming here. For reading, you read into LMD, for writing it will come from B and IR is forwarded, ALUOut is forwarded. And one thing, here this ALUOut and the cond signals are that generated in MEM, are fed back to the IF stage because here at this stage you have known for sure that whether it is a branch instruction or not, and whether the condition is true or false.

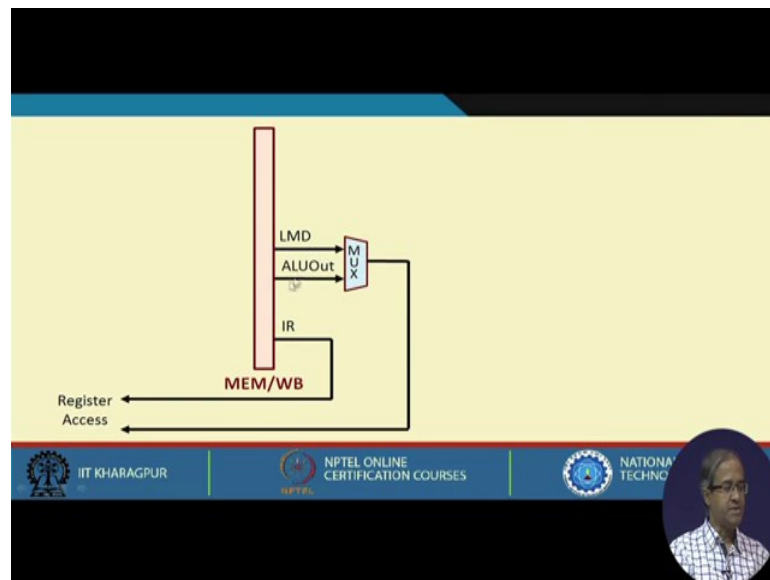
So, you send them to the IF stage indicating the condition of the branch instruction and what is the target address of the branch so that PC can be updated accordingly.

(Refer Slide Time: 15:12)



Lastly in the in WB stage you are doing some write into the register banks depending on R-R, R-M or LOAD instruction.

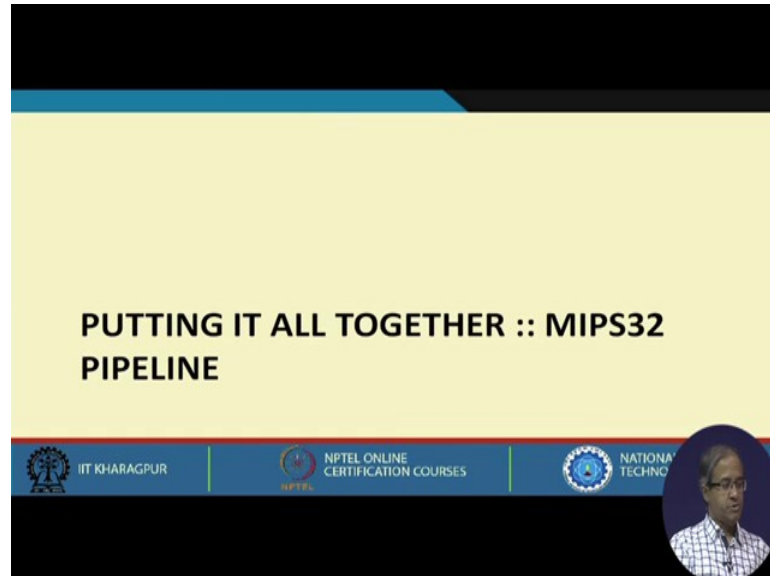
(Refer Slide Time: 15:35)



It is fairly simple, it takes the LMD and ALUOut from here and IR. Why is this IR forwarded? Because you see depending on the instruction you need this fields rd or rt, because they have to be written into the register. So, these signals will have to go back to the register bank. You see here these signals were coming from the WB stage. The register address was coming rd or rt; here rd is shown, but it can be rt also, and also the

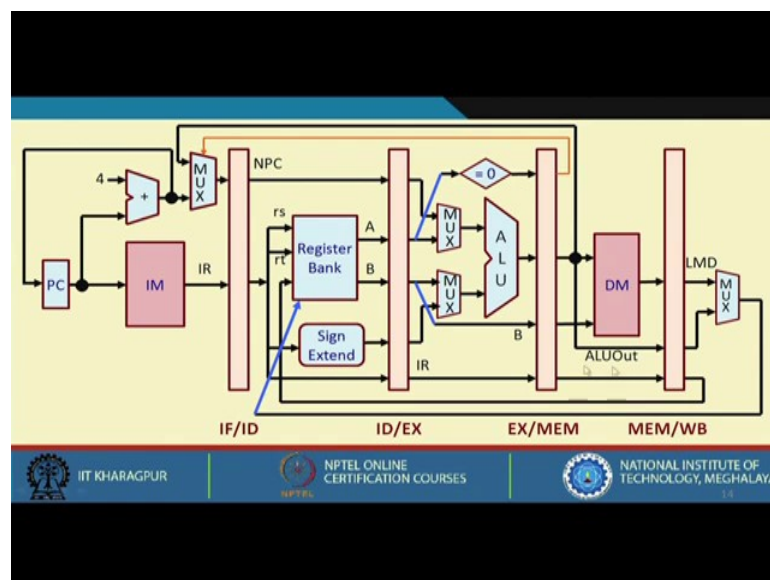
data to be written. Here those data have been fed back to ID stage, and this MUX selects either LMD or ALUOut depending on the type of instruction.

(Refer Slide Time: 16:44)



The control circuit will be selecting this multiplexer. Putting all the things together the MIPS pipeline looks something like this.

(Refer Slide Time: 16:49)



Well this is a little complex; I have tried to fit it into a slide. This is your IF. I have shown this PC from here, but actually this can be from the output of the MUX also. You can either do it always + 4, you load it or you can make it as default environment, it does not

matter really. Earlier I have shown that this connection coming from here, but here I am showing that directly $PC = PC + 4$ you are doing here, but if it is a branch later it is known then from the output of the MUX you have to make some changes. For that change this connection, it has to be taken from the output of the MUX. Here just for simplicity I have not shown the exact connection. This is our instruction memory and this is your ID stage; register bank, two read and one write, and sign extension. These rs and rt are for reading and the third one for writing, and this is the data to be written to the register. This is the EX stage; MUX selects either NPC for branch instruction or A for other instruction. They depend on instruction, MUX will be selecting one of the inputs, it goes here. So, $cond$ is getting generated here, this $cond$ will be selecting this MUX.

Actually this will be fed here; data memory MUX. This is how the total MIPS32 pipeline implementation looks like. You see it is not at all complex; it is very similar to our earlier non-pipelined diagram with some minor changes. We shall now look at some other issues or problems called hazards. We shall be exploring the various hazard situations and try to come up with some solutions.

So, let us stop for today in this lecture. We shall be continuing with the hazard detection and avoidance in our subsequent lectures.

Thank You.