Hardware Modeling using Verilog

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

Lecture - 39
Pipeline Implementation of a Processor (Part 3)

So, now in this lecture, we shall be starting from where we left in the last lecture and we shall be looking at what kind of modifications we need in the micro operations and also in the data path, if we want to pipeline or MIPS 32 means instruction set architecture implementation now means one thing is happening naturally because earlier we talked about 5 different stages of execution or steps of execution.

So, for the pipeline implementation also, we shall be retaining that same definition for the pipeline stages the same 5 steps will be converting into pipeline stages.

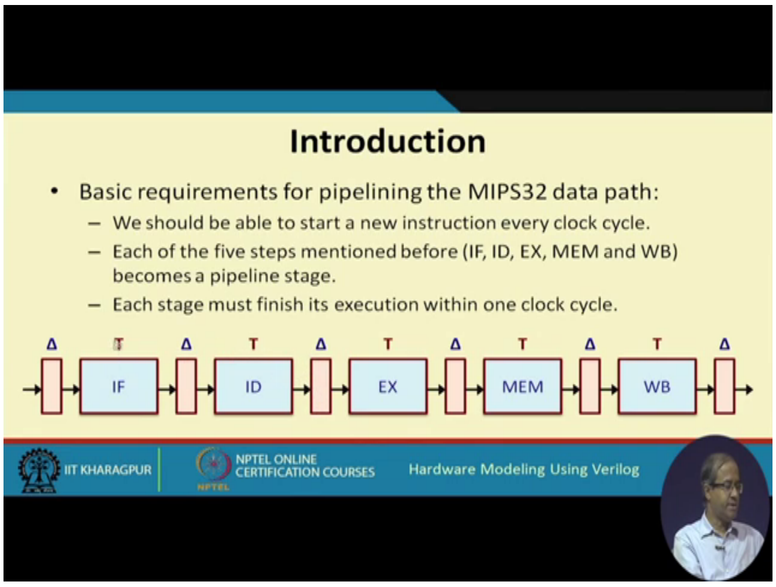
(Refer Slide Time: 01:08)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES NPTEL

Lecture 39: PIPELINE IMPLEMENTATION OF A PROCESSOR (PART 3)

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(Refer Slide Time: 01:12)


Introduction

- Basic requirements for pipelining the MIPS32 data path:
  - We should be able to start a new instruction every clock cycle.
  - Each of the five steps mentioned before (IF, ID, EX, MEM and WB) becomes a pipeline stage.
  - Each stage must finish its execution within one clock cycle.

Δ
T
IF
ID
EX
MEM
WB
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, this is the title of our talk. So, first we look at what are the basic requirements for pipelining the data path that we have seen in the last lecture. So, what are the things to be done?

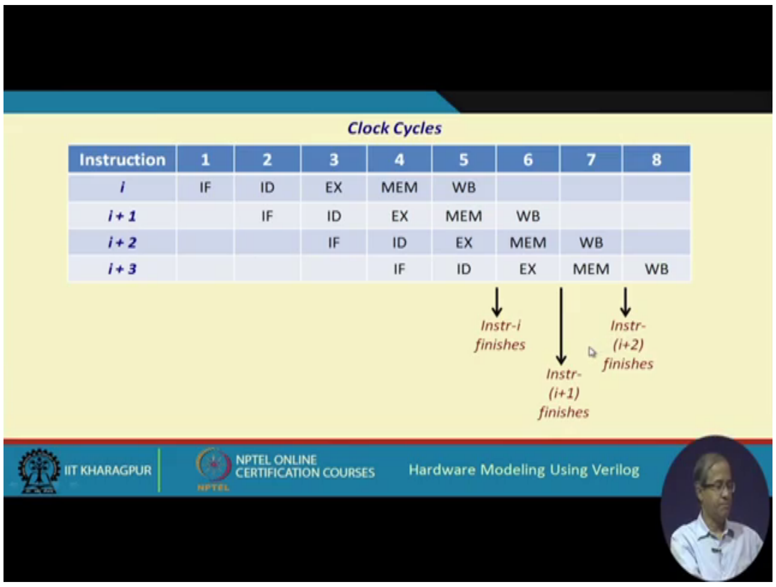
Well, in a pipeline what we have seen earlier through some examples; the main motivation for having a pipeline is that our input data is arriving continuously once every cycle. So, we should be able to feed one new data every cycle. Now, in this case, we are talking about instructions our instructions is like our data in every cycle, a new instruction is coming a new instruction is fetched; this is how the pipeline will be executing and this is called a this is referred to as instruction pipeline because this is a pipeline where instructions are executing and instructions are moving from one stage to the other IF, ID, EX, MEM, WP, fine.

So, the one requirement is that we should be able to start a new instruction every cycle because otherwise this pipelining will not give you the desired advantage and as I said the 5 steps that we had mentioned these 5 steps; we will make them as 5 pipeline stages IF, ID, EX, MEM and WB.

Now, of course, one thing is important you see there is a global clock with clock the data and the results are moving. So, every stage should be able to finish its execution within a clock cycle. So, that is another requirement each stage must finish its execution within a clock cycle.

So, our overall pipeline will look something like this. This will be the 5 stages and there will be pipe, there will be pipeline latches separating the stage just like we saw earlier for isolation, we need this latches because when the first instruction as move from IF to ID, its temporary data are stored in this latch. So, ID can very peacefully do the instruction decoding because these data are stored in the latch and mean time; this second instruction can be fetched in IF, but if you do not have this latch while IF is going on this data will this will keep changing and so the process of decoding might get disturbed just for isolation we need the latches.

(Refer Slide Time: 04:08)



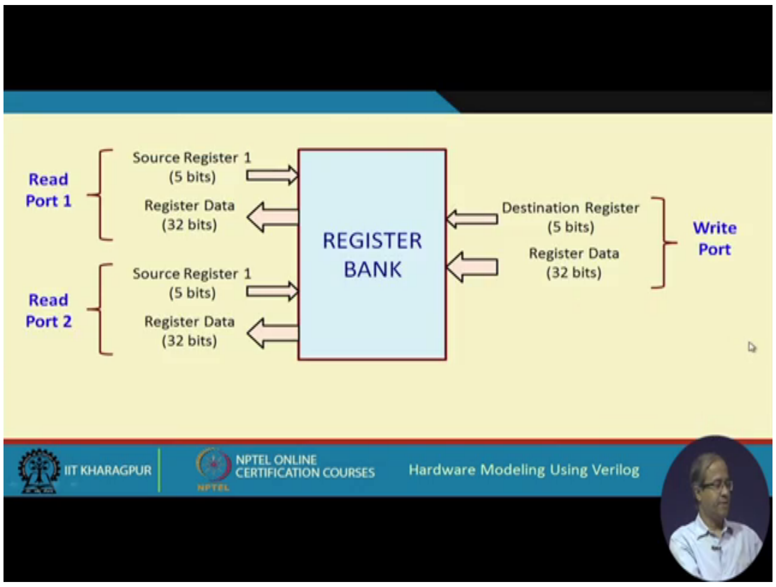
Clock Cycles

Instruction	1	2	3	4	5	6	7	8
i	IF	ID	EX	MEM	WB			
i+1		IF	ID	EX	MEM	WB		
i+2			IF	ID	EX	MEM	WB	
i+3				IF	ID	EX	MEM	WB

Now, in instruction pipeline the advantage that you get pictorially depicted like this you see in this direction we show the time the clock cycles, let us say our instruction number I is coming let us start with clock cycle number one. So, it is fetched decoded executed MEM and WB, it is finished. Now in pipelining what we do when the first instruction I mean instruction, I moves from IF to ID. The next instruction in sequence it can start IF. So, in time step 2 the first instruction is in ID the next instruction is in IF in third step the first instruction has moved to x second in ID and third in IF in this way you see fourth step MEM, IX, ID, IF fifth clock cycle WB, MEM, EX, ID.

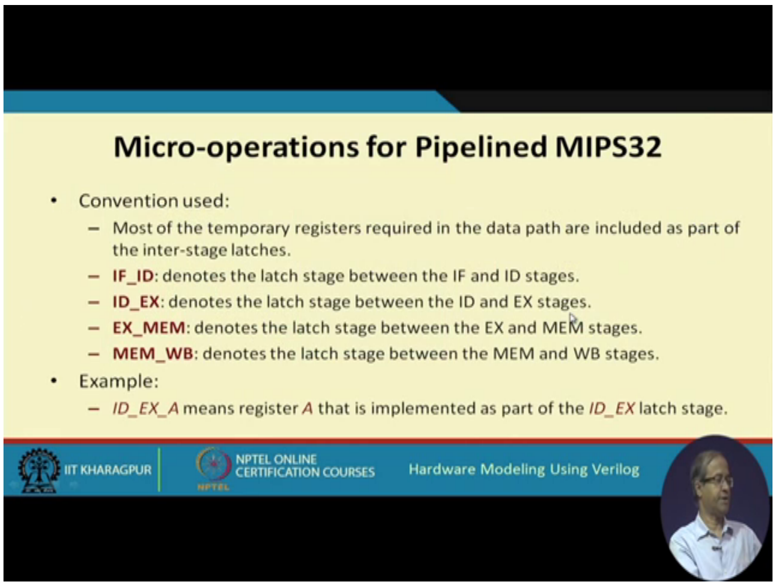
Now, say after this WB the first instruction instead of now I will finish. So, it will finish here in after 5 cycles and after 6 cycles, I plus 1 will finish because WB is finished here and after seven cycles I plus 2 is finish and so on. So, here the ideas that you initially give some time for the pipe to fill up and after it gets filled up, you will be getting one instruction completed per clock cycle, this is the advantage that we gain in pipelining, we also saw the example earlier, if there are 5 stages, you can in the ideal case expect 5 times speed up, but of course, in instruction pipeline there are other constraints well talk about it later that limit the speed up it may not be exactly 5 in fact, much less than that and I plus 3 here.

(Refer Slide Time: 06:21)


Read Port 1
Source Register 1 (5 bits)
Register Data (32 bits)
Read Port 2
Source Register 1 (5 bits)
Register Data (32 bits)
REGISTER BANK
Destination Register (5 bits)
Register Data (32 bits)
Write Port
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

And this is a picture; you also saw earlier for this case also, we need a register bank that will be having 2 read ports and one right port. So, there will be 32 registers that is why the source and destination registers are 5 bits each and each register is 32 bits long. So, the register data are all 32 bits, right.

(Refer Slide Time: 06:49)



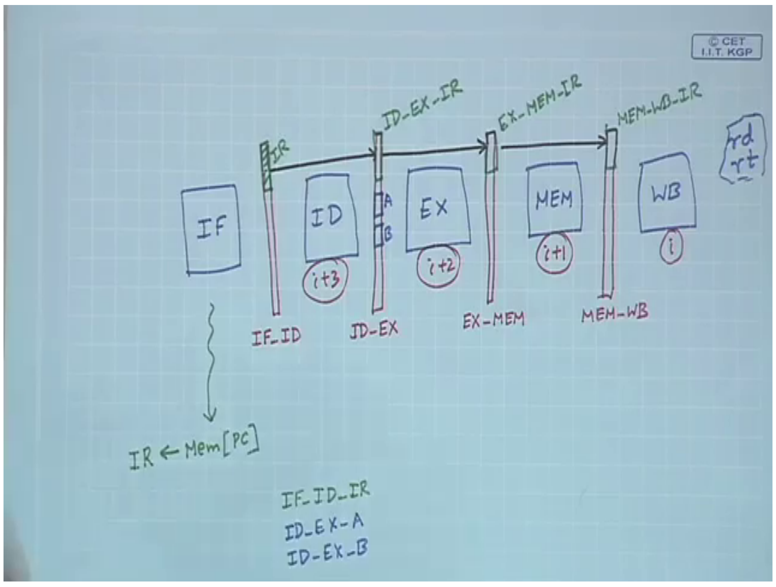
Micro-operations for Pipelined MIPS32

- Convention used:
  - Most 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 register *A* that is implemented as part of the *ID_EX* latch stage.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Now, let us try to modify the micro operations for the pipelined MIPS 32 version. Now the first step is that we need to follow some conventions for this because the conventions that we are talking about it with respect to the naming conventions and we will be consistent with name; that means, whatever we shown the slides the same convention will be following when you show the Verilog codes.

(Refer Slide Time: 07:22)



Now, the idea is as follows. You see you have this 5 stages instruction fetch instruction decode execution MEM and write back WB. Now what I am saying is that you see; there will before and after, but the most important latches are those are between these stages. So, for our interest we need these 4 inter stage latches, we give some names to these latch stages, we call this stage as IF underscore ID; that means, a latch stage that is sitting between IF and ID this latch stage we call as ID underscore EX, we call this EX underscore MEM and we call this MEM underscore WB. So, this is the conventional, we following the latch stage, we will be representing using this underscore notation between which pair of stages this latch stage is, fine.

So, this is the convention that I have just talked about IF, ID; ID, EX; EX, MEM and MEM, WB and another thing the temporary registers that we have talked about let us take an example. Now in the in the IF stage just look at the IF stage in the IF stage, you recall there was a micro operation which says that instruction register was loaded with memory location whose address was in PC.

So, the data must be stored in some register IR. So, where is IR.? So, IR we read the some separate register, inside IR is actually know actually just like in the example that we showed earlier that all temporary registers will be part of the latches register stages let us say this one this itself can be IR. So, and this register actually will be referring to as IF underscore ID underscore IR, this is the kind of naming convention we will be following.

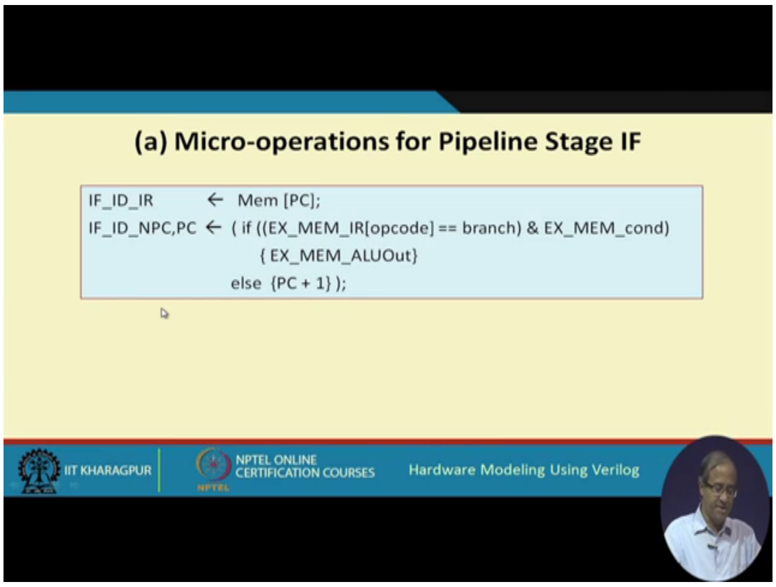
And in case this register is also forwarded here let us say maybe this IR will require later also. So, here also there will be an IR and this IR will be calling as ID underscore EX underscore IR and this same IR will also be forwarding here. So, here will be calling it, there will be an IR register here we calling it EX, MEM, IR and again this will also be forwarded here this will calling MEM, WB.

Now, the reason I have taking the example of IR is there is a reason for good reason for this you see this IR is just one example there; there will be there will be there will be many other registers which will also be just handle is same way. For example, in the instruction decode when you pre fetch the register bank there will be one register called A, another register called B, there will be similarly named IF, this will be ID EX, this will be ID EX A and ID EX B.

Now, the reason that we have to forward it like this you see instruction register will contain the entire instruction format all the source register destination register for some of the instructions you need to take the decision only at the end, let us say ride back in for some instructions you are writing in to register R d for some other instructions, you are writing into R t. So, both the values of R t and R t should be available to you that is present in this instruction IR and you need to forward it because when instruction, I has reached WB, let us say here you have instruction number I, your next instruction is here next instruction is here and I plus 3 is here and so on.

So, whatever IR is stored here that is the IR of instruction I and whatever IR is stored here that is the IR of instruction I plus 1, this is the IR of instruction I plus 2 and this will be the IR of instruction I plus 3. So, this explains why; we need to forward the same register across stages. So, many numbers of times because there are so many instructions moving and each of them will be needing their own IR their instructions are different. So, their IR will also different that is why.

(Refer Slide Time: 13:10)


(a) Micro-operations for Pipeline Stage IF

```
IF_ID_IR        ←  Mem [PC];
IF_ID_NPC,PC ←  ( if ((EX_MEM_IR[opcode] == branch) & EX_MEM_cond)
                        { EX_MEM_ALUOut}
                  else  {PC + 1} );
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

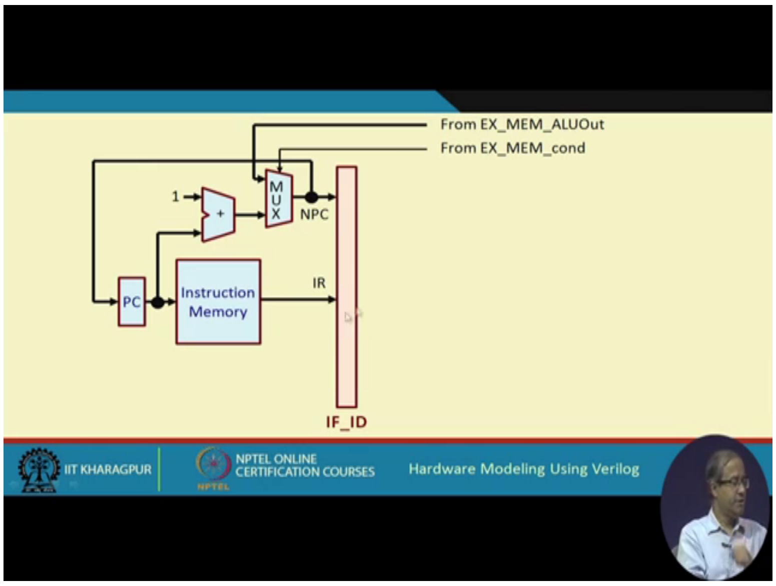
So, now let us straight away look at the micro operations for the different stages well assuming that we are having a pipeline implementation first the instruction fetch you see here whatever we are showing we shall be directly translating them into Verilog code later, we are fetching the next instruction MEM PC, earlier we mentioned that we are storing it in IR, but now we are saying it will be IF underscore ID underscore IR this IR will be has part of the IF latch.

Then we check whether the opcode is a branch and the branch condition is true you see a branch instruction is fetched decoded, but the branch condition will be known only when it has reached the EX stage, just you see this diagram once more see an instruction a branch instruction. So, once it reaches the EX stage, only then you know that is a branch instruction and the values of the conned and the next instruction address will be stored in this buffer.

So, if you decide that there is a branch. So, the next instruction has to fetched from that address. So, there has to be some kind of a feedback from this back to here next instruction fetch will depend on the outcome of these conditions of the branch right that is reflected by this micro operation you see here it is seeing if the instruction registered in the EX MEM you see EX MEM is this one EX MEM opcode was a branch and the corresponding condition branch condition was true then only branch will take place you see.

So, if EX MEM opcode is branch and EX MEMs branch condition was true then whatever branch targeted this if calculated in ALU out EX MEM ALU out that will be assigned both to IF ID NPC and also to PC else, if this branch it is not a branch out the branch condition is not true then simply PC plus 1 will be assigned to these 2, fine.

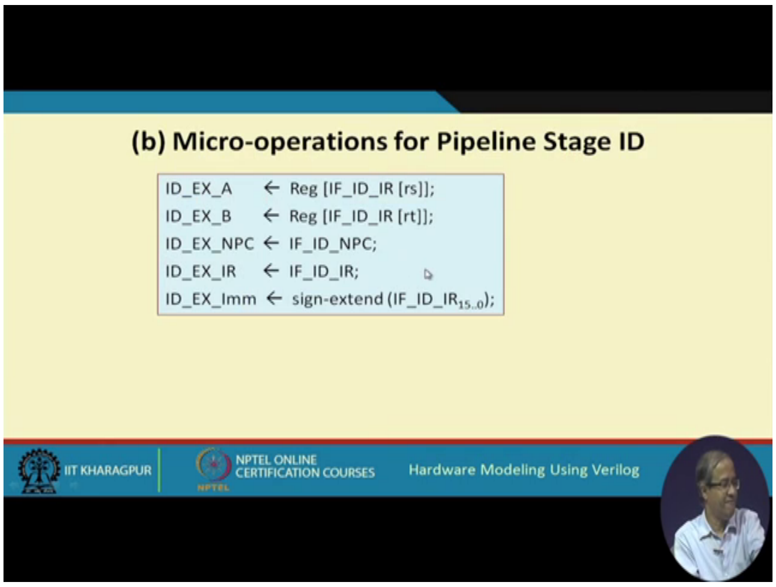
(Refer Slide Time: 15:59)



So, in terms of hardware the instruction fetch stage will look like this, this will be your IF ID stage instruction you know IR will be part of this NPC will also be part of this and as I said from MEM out stage EX MEM stage these 2 signals are coming, right. So, you are just calculating the NPC accordingly and after calculating NPC that same value also goes to PC, you see previous PC and NPC both are getting the same values will else PC plus 1 if the branch condition is if cond is not true, then their lower input will be selected from the multiplex that is PC plus 1 that PC plus 1 will go back to PC plus 1.

Now, see one thing is true here see for a non pipeline implementation we were storing the next address in NPC much later NPC was copied to PC, but in a pipeline we cannot afford to do that why in every cycle we are fetching a new instruction. So, we will have to continuously update the PC in every cycle that is why we made this change here in every cycle we are updating also the PC, right. So, the PC is getting updated directly here, fine.

(Refer Slide Time: 17:37).



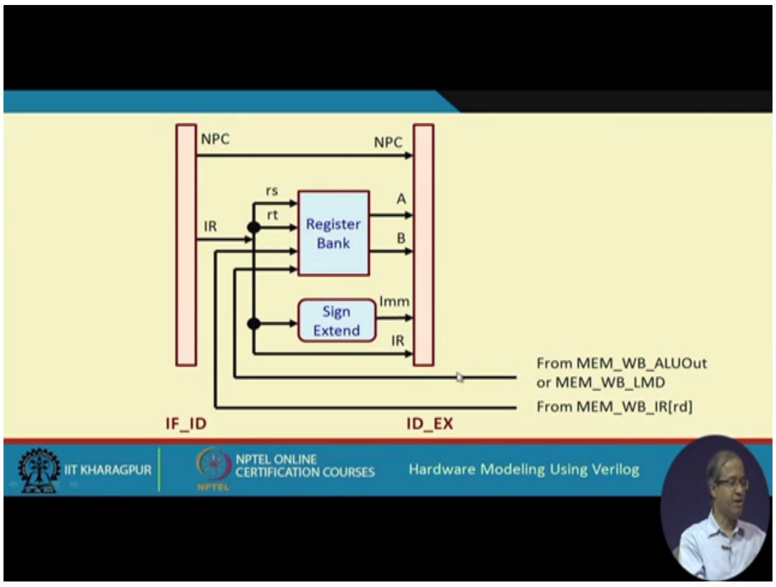
(b) Micro-operations for Pipeline Stage ID

```
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);
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

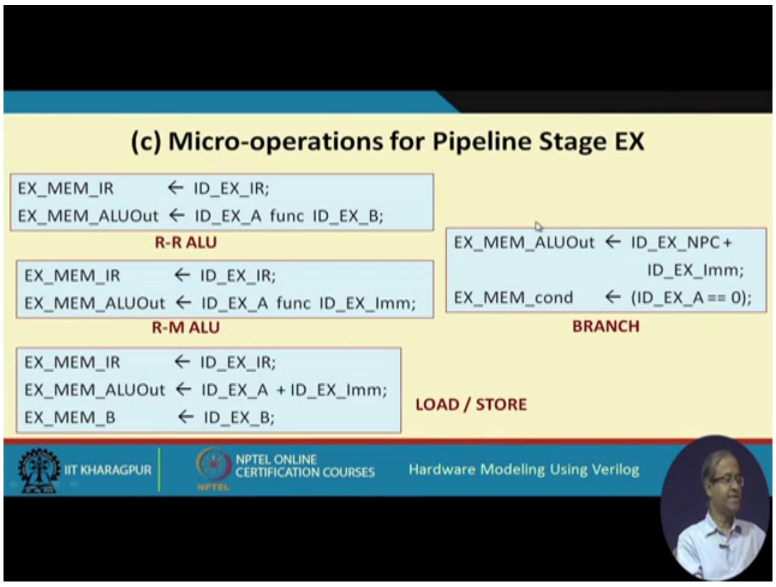
Now, let us move to the ID well ID is very similar only the variable naming convention has changed see Reg earlier we said Reg R s, but now we are saying IF ID IR R s of that IF ID IR R t the 2 registers are pre fetched they will be stored in ID EX A and ID EX B similarly NPC is just copied IR is just copied as I said and Imm is sign extended. So, IF, ID, IR; this bits will be a sign extended to ID EX Imm

(Refer Slide Time: 18:20)


NPC
NPC
rs
rt
IR
Register Bank
A
B
Imm
Sign Extend
IR
From MEM_WB_ALUOut or MEM_WB_LMD
From MEM_WB_IR[rd]
IF_ID
ID_EX
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, in terms of the hardware, it will look like this register bank. So, NPC will be copied this IR will be copied that Imm part of the IR will be sign extended to get Imm; R s and R t, there will be pre fetch register bank will go to a and b and from MEM, WB, ALU out or LMD, the data will be coming for writing the register bank and from MEM WB IR the target will be coming this will be either R r d ot R t. So, here we although written R d, but this can be R t also depending on the type of instruction this is how it will be in this stage.

(Refer Slide Time: 19:08)



(c) Micro-operations for Pipeline Stage EX

```
EX_MEM_IR      ← ID_EX_IR;
EX_MEM_ALUOut  ← ID_EX_A func ID_EX_B;
```
R-R ALU

```
EX_MEM_IR      ← ID_EX_IR;
EX_MEM_ALUOut  ← ID_EX_A func ID_EX_Imm;
```
R-M ALU

```
EX_MEM_IR      ← ID_EX_IR;
EX_MEM_ALUOut  ← ID_EX_A + ID_EX_Imm;
EX_MEM_B       ← ID_EX_B;
```
LOAD / STORE

```
EX_MEM_ALUOut  ← ID_EX_NPC +
                         ID_EX_Imm;
EX_MEM_cond    ← (ID_EX_A == 0);
```
BRANCH

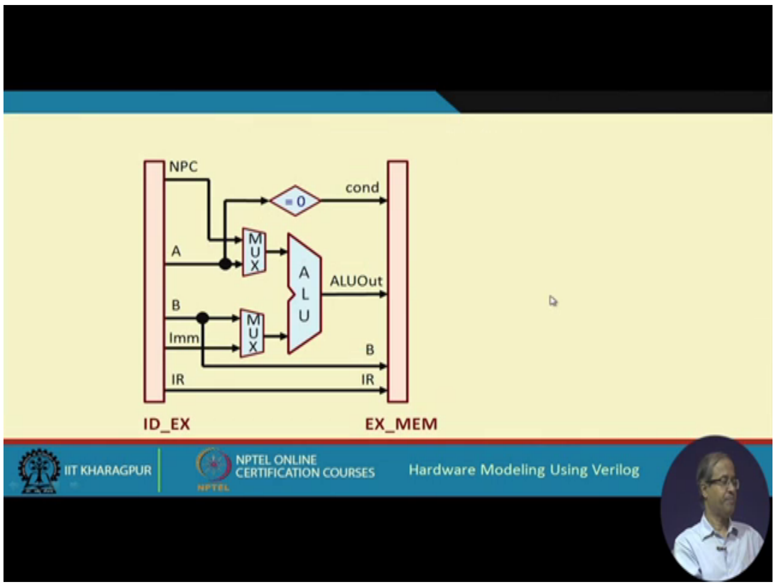
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Similarly, for the EX stage depending on the instruction type micro operations will vary for register ALU operation first this IR has to be copied across every stage IR is simply copied and A the function whatever add subtract multiply be restored in ALU out, but these variables are all prefixed by the pipeline latch number similarly raised memory ALU IR is copied and here instead of B it is Imm.

For load store IR is copied this A is added to immediate data to compute the effective address and B is also copied because for store instruction, this B will be stored. So, B also you need to copy forward. So, that in the MEM stage you can execute the store instruction and for branch you add NPC and Imm because NPC was coming and you check the condition A equal to equal to 0 that condition can be either true or false that is stored in cond EX MEM cond.

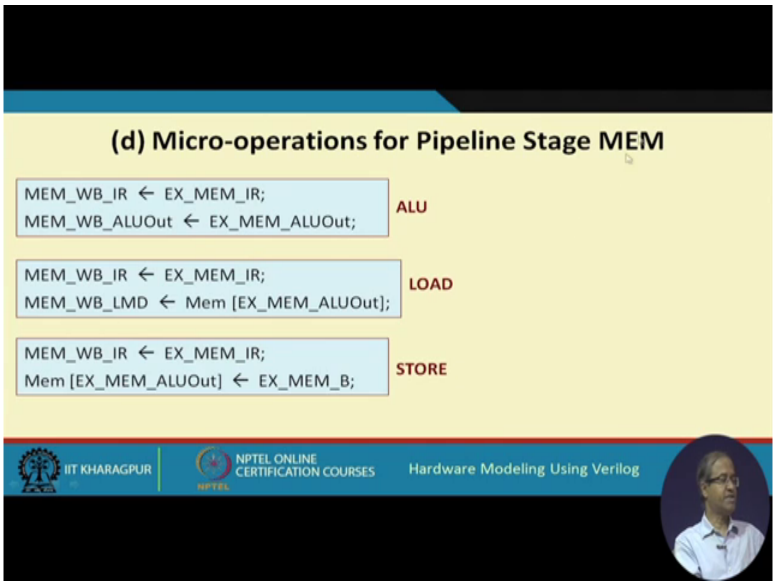
You recall this EX MEM cond actually is being used here you see earlier this EX MEM cond, I showed here in the fetch instruction fetch stage this is coming from there.

(Refer Slide Time: 20:39)


NPC
A
B
Imm
IR
=0
cond
MUX
MUX
ALU
ALUOut
B
IR
ID_EX
EX_MEM
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, the EX stage the hardware will look as simple as this ALU with some multiplexer, it can be either come from NPC or these are all part of this latch NPC A B Imm IR. So, all these registers are here and in the output, here cond is there ALU out is there B is there IR is there.

(Refer Slide Time: 21:01)



(d) Micro-operations for Pipeline Stage MEM

```
MEM_WB_IR ← EX_MEM_IR;
MEM_WB_ALUOut ← EX_MEM_ALUOut;
```
ALU

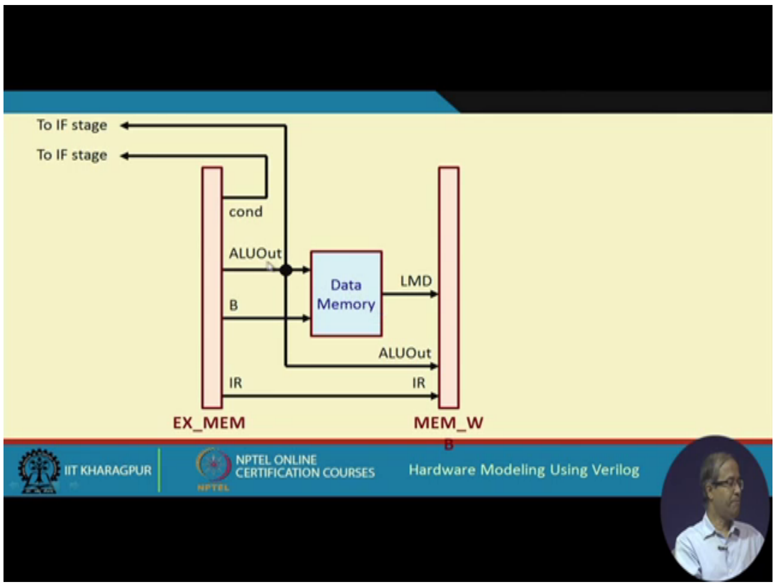
```
MEM_WB_IR ← EX_MEM_IR;
MEM_WB_LMD ← Mem [EX_MEM_ALUOut];
```
LOAD

```
MEM_WB_IR ← EX_MEM_IR;
Mem [EX_MEM_ALUOut] ← EX_MEM_B;
```
STORE

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

Now, let us come to MEM for ALU operation again IR is forwarded and also ALU out is forwarded nothing else to do in MEM simply forward the data because you will be using them in the next stage load IR is forwarded and do a memory read. So, ALU out contains the address you read it and store in LMD for store you see B was forwarded earlier, this B is getting written to memory ALU out is the address right. So, this will be your MEM.

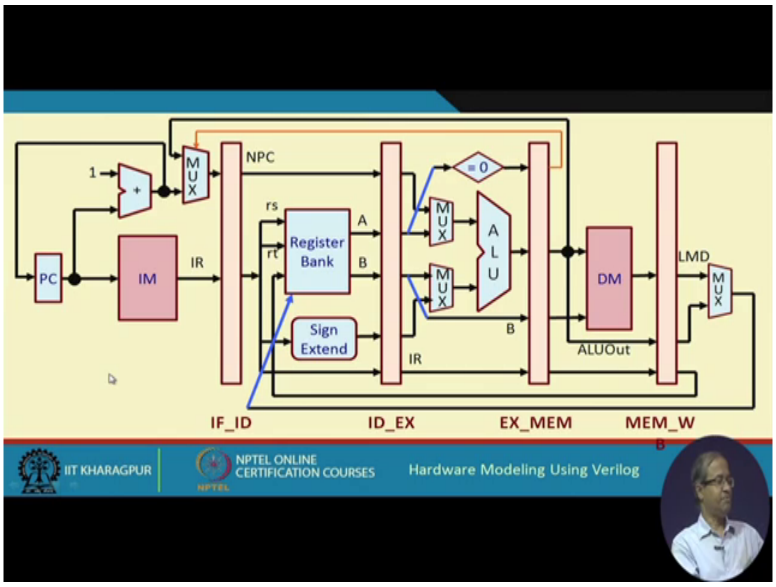
(Refer Slide Time: 21:38)



ALU out is the address and b is for write and LMD is for read and from this cond end this ALU out going to the previous stage, as we showed earlier and IR is again forwarded and also ALU is forwarded to the next stage.

Lastly for WB you see here both LMD and ALU out are used that is why LMD is forwarded LMD is there ALU out is there and also IR is there because IR you require R d or R t. So, if register ALU operation then ALU out will be stored in R d that register for register memory it will be stored in R t for load LMD will be stored in R t fine. So, here your WB is nothing, but a multiplexer where we will be selection either LMD and ALU out depending on a control signal and it will be going to your means ID stage. So, it will this IR will contain which register number and this will contain what is the data.

(Refer Slide Time: 22:57)

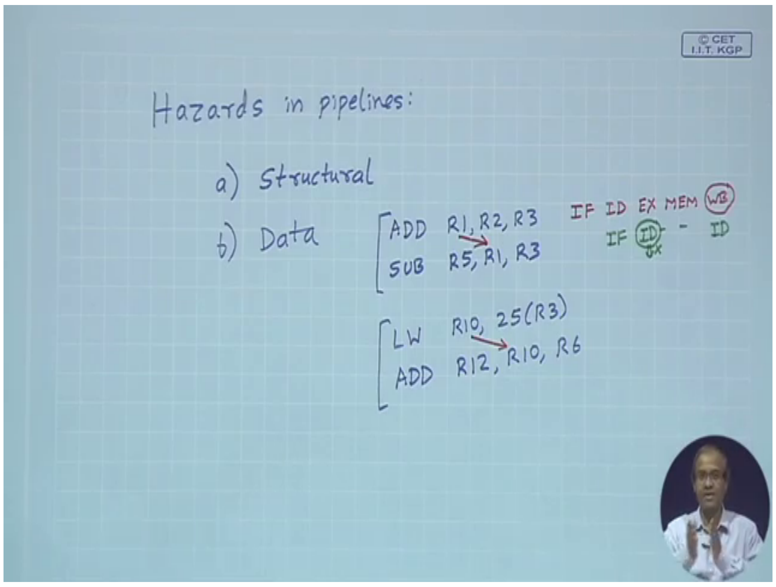


So, if you combine everything together whatever we have discussed so far you see the picture looks like this just see the simplicity of the thing the whole processor pipeline is fitting within a single slide; let me just make it bigger this, yeah, you see this is your instruction fetch decode execute memory WB just let us proceed with one just example let us say add R 1, R 2, R 3. So, what happens in IF PC contents the address of the instruction it is fetched. So, the instruction is loaded in IR, here the 2 operands are pre fetched A and B, the 2 source operands in for EX, this A is selected, B is selected, they are added result will be stored in ALU out for memory, it does nothing ALU out is simply forwarded and for WB this ALU out will be written back into the register bank and this IR will contain which register number.

You see this is the simple data path that we have shown; I just one thing; I wanted to clarify here you see; here I am not trying to teach you what is the processor what is a pipeline processor how to design a pipeline processor basically I am taking it as an illustrative example to tell you how we can model complex pipelines in Verilog. Now well because we have taken the example of an instruction pipeline let me tell you a few things.

Well, for a pipeline you may imagine that well if I have a k stage pipeline, I can have a speedup of k potentially when there are large number of data coming one after the other, but when you talk about instructions there are a number of other problems that may come there are situations which are called hazards.

(Refer Slide Time: 25:13)



Hazards in pipelines:

a) Structural

b) Data

```
[ADD  R1, R2, R3        IF ID EX MEM WB
[SUB  R5, R1, R3           IF ID  -  ID
                              EX

[LW   R10, 25(R3)
[ADD  R12, R10, R6
```

© CET
I.I.T. KGP

These hazards can happen in pipelines first kind of hazard can be structural hazard. So, what is the structural hazard structural hazard means 2 instructions which are already there in the pipeline; they are in 2 different stages, but they are trying to use the same hardware resource if there is a single copy of the hardware resource they cannot access it together. So, one of them will have to wait and the other can proceed this is an example where you may have to insert A. So, called stall cycle in a pipeline; that means, one clock cycle may get wasted.

Like you see an example here in this pipeline in the IF you are fetching instructions from memory and in MEM for load and store instruction; also you are you are accessing memory suppose if you had a single memory then you could not do; this 2 things together that is you see I have separated them out instruction memory and data memory, I have separated out the 2 memory so that this kind of structural hazard can be removed. So, instruction will be stored in one memory data will be stored in another memory something like that we can assume.

There are other more important kind of hazards like there is something called data hazard data hazard can arise due to instruction dependency let us take just some examples let us say I have an just add R 1, R 2, R 3 followed by a instruction let us say subtract let us say R 5, R 1, R 3, this is one example let us take another example load R 10, let us say 25 R 3, add R 12, R 10, R 6, this is another example.

So, you see one thing here the first instruction is generating R 1 which is been used in the second instruction, here it is generating R 10, it is used in the second instruction. So, you think like this first instruction add this was going through instruction fetch instruction decode then execute MEM then write back. So, under normal circumstance this R 1 will be written result in written only during WB.

But for the next instruction, you see sub instruction here fetching will be done here and we will be trying to do instruction decode and register pre-fetching here, but because R one is not yet ready. So, here you will not be able to fetch the correct register value. So, you will have to wait maybe possibly you can do your instruction decoding and pre-fetching here while this is being written. So, 2 cycles you may have to stall in between.

Same is the case for here you are loading an instruction into memory this will be done only in the MEM stage and in the next case you are using that value R 10. So, again R 10 written in WB again something similar will happen well there are methods to reduce the impact of this kind of hazard there is something called data forwarding and other things, but this is something you have to remember later on when you say some examples on the Verilog implementation we will see that for sample programs, we have to deliberately insert some dummy instructions in between such that the result is correct otherwise this kind of hazard will take place and the next instruction will try to read some wrong value instead of the value which is being computed by the previous statement.

And the third kind of situation is called control hazard this arises because of branch instructions like you see a branch instruction has entered the pipe. Now you take the decision of the branch only after EX, right. So, when the branch instruction has entered the EX stage already 2 other instructions are already in the pipe. So, when you decide that you have to branch. So, those 2 instructions have to be discarded. So, these are some reasons why; you cannot utilize the pipeline to the fullest possible extent and there can be some stall cycles, but again, I am telling you in this class in this sequence of lecture our objective is not to teach you computer architecture, but rather we want to teach you how to model this kind of pipelines in Verilog.

So, all these things we are just assuming that these are required that us why you are doing it, we are not going into the detail or we are not talking about any analysis on that. So, with this we come to the end of this lecture from the next lecture onward, we will be actually looking at the Verilog implementation of this processor that we have discussed and talked about so far.

Thank you.