

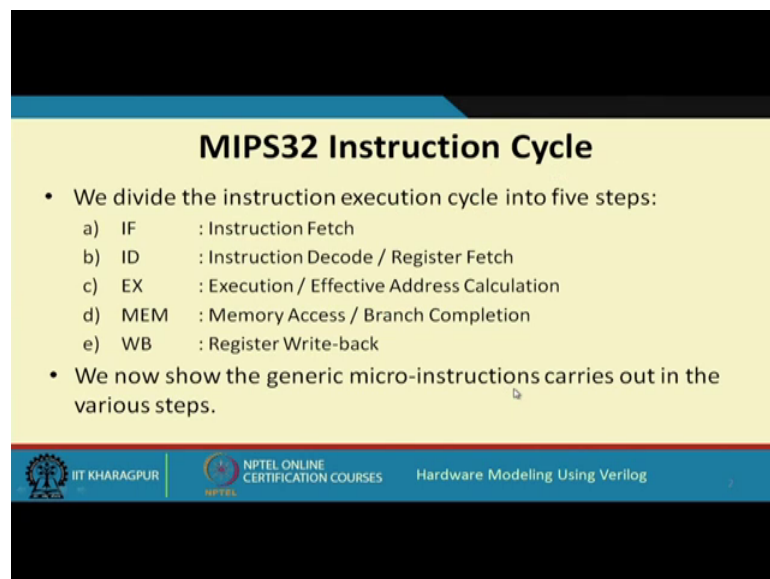
Hardware Modeling using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 38
Pipeline Implementation Of A Processor (Part 2)

So, in the last lecture we started our discussion on the MIPS 32 instruction set architecture; a small subset of it and was talking about the instructions that we are actually going to implement as part of these lectures.

Now, in the present lecture we shall try to see what are the steps involved in instruction execution. We shall not be talking about pipelining now; we shall be looking into pipelining later. Now, you forget pipelining, but what we are trying to understand and learn is that; for any instruction it can be add, it can be load, it can be branch; what are the steps that need to be carried out inside the processor so as to execute or fulfil the functionality what is actually required of the instruction. So, this is our part two of the discussion on this.

(Refer Slide Time: 01:22)



MIPS32 Instruction Cycle

- We divide the instruction execution cycle into five steps:
 - a) IF : Instruction Fetch
 - b) ID : Instruction Decode / Register Fetch
 - c) EX : Execution / Effective Address Calculation
 - d) MEM : Memory Access / Branch Completion
 - e) WB : Register Write-back
- We now show the generic micro-instructions carries out in the various steps.

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

Let us come straight to a breakup of the instruction cycle; now talking about a instruction cycle, instruction cycle refers to a time period that is required for the execution of a complete instruction. Now, here we are saying because the processor or the instruction

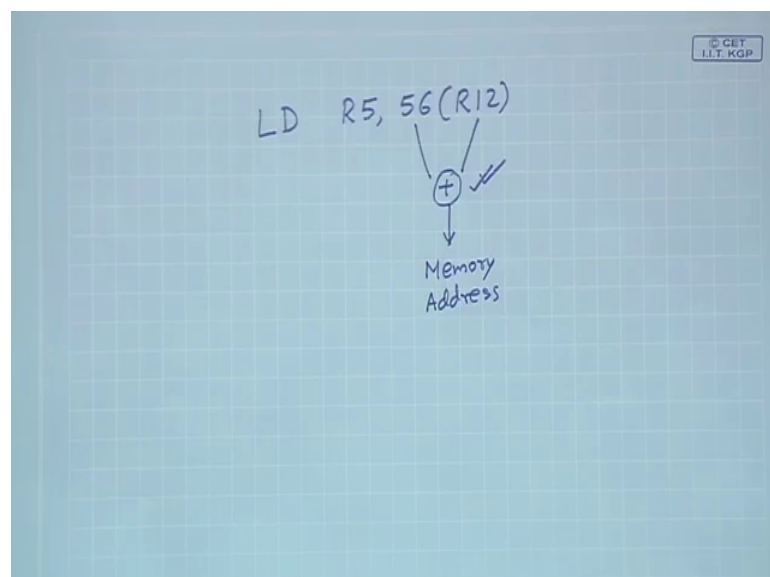
set that we are trying to design and implement is very simple; this instruction cycle can be very regularly divided into 5 different you can say sub cycles or steps.

So, what we are saying is that; the total instruction execution cycle we divide into these five steps. So, what are this 5 steps? First step is called Instruction Fetch; so, here we are fetching an instruction from the memory. So, we know that the program counter the PC register already contains the address of the next instruction. So, simply whatever is there in PC from that memory location we read that is instruction fetch.

Then second step is instruction decode; here we are trying to decode the Opcode and find out what kind of instruction it is? Is it add, subtract, multiply or what. And in parallel while decoding is going on as I have said; we also do some register prefetching. Well we assuming that the instruction will be requiring both source registers we will be prefetching the registers; not only that assuming that there will be a 16 bit immediate data, we will take that last sixteen bit of that instruction and we will be doing a sign extension to 32 bits. These things we are doing in anticipation; we really do not know whether these things will actually be required or not it will be known only after decoding of the instruction is complete.

And the third step is execution where actually an arithmetic logic unit is used. Here we execute the instruction or for some instructions, we can we have to compute the effective address.

(Refer Slide Time: 03:56)



Well you think of a load instruction; the load instruction that we talked about was like this; load let us say R 5, 56, R 12. So, the value of 56 and R 12; they will be added and the result of the addition will be your memory address; from where the data will be loaded into R 5.

So, you see you require an addition step here; in the load instruction no other arithmetic calculations required, only this addition to compute the effective address or the operand address. So, that it is what is mentioned here in the E X stage; for such load and store instructions, we can also do the effective address calculation. Then in MEM; we can actually do the memory access read and write from memory and also for branch instruction; we can also decide whether to branch or not and the last stage W B; it is register write back.

Let us say I have an add R 1, R 2, R 3 instruction; so, R 2 and R 3 will be added; the result will be stored in R 1. So, the result stored in R 1; this will be taking place in the W B stage; last stage W B stands for Write-back; Register Write-back.

(Refer Slide Time: 05:40)

(a) IF : Instruction Fetch

- Here the instruction pointed to by *PC* is fetched from memory, and also the next value of *PC* is computed.
 - Every MIPS32 instruction is of 32 bits.
 - Every memory word is of 32 bits and has a unique address.
 - For a branch instruction, new value of the *PC* may be the target address. So *PC* is not updated in this stage; new value is stored in a register *NPC*.

IF: $IR \leftarrow Mem [PC];$
 $NPC \leftarrow PC + 1;$

For byte addressable memory, PC has to be incremented by 4.

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

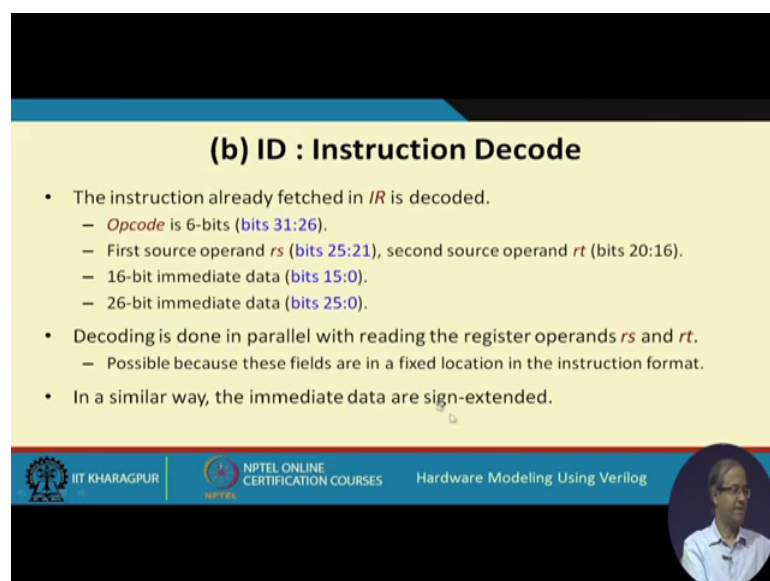
And a different steps that are going on; even within each of these steps, there are several sub steps these are called micro instructions and micro operations. We shall be showing you what are the micro operations that are required in the different steps you will be able to understand it.

Let us start with instruction fetch; here I have said that the instruction pointed to by PC; PC always contains the address of the next instruction to be executed. So, whatever PC contains; you fetch that word from memory and store it in an internal register it is called an instruction register you see it later, but this fetch is going on. And we are assuming this already we have mentioned earlier that every instruction is 32 bits long; also every memory word is 32 bits and has a unique address so that after an instruction has been fetched, you will have to increase PC by 1 to point it to the next instruction.

So, what you do? This is the instruction fetch MEM of PC; this is memory just like accessing array; something similar to that. You are accessing memory; this address is in PC and you are storing into a register called I R; this I R stands for instruction register. And you increase the PC and do not store it back in PC because for jump instruction; the target address can be something else you do not know it yet. So, you use another register called new PC; NPC store it temporarily in NPC; this is what we do during the instruction fetch stage.

Now, just one thing for some memory systems; the memory words are byte oriented means every byte has a unique address. So, if we talk about 32 bit instructions; so, we will have to increase PC by 4 to go to the next instruction because there are 4 bytes in every instruction, but for our example; we are considering word addressable to keep it simple.

(Refer Slide Time: 07:47)



(b) ID : Instruction Decode

- The instruction already fetched in *IR* is decoded.
 - *Opcode* is 6-bits (bits 31:26).
 - First source operand *rs* (bits 25:21), second source operand *rt* (bits 20:16).
 - 16-bit immediate data (bits 15:0).
 - 26-bit immediate data (bits 25:0).
- Decoding is done in parallel with reading the register operands *rs* and *rt*.
 - Possible because these fields are in a fixed location in the instruction format.
- In a similar way, the immediate data are sign-extended.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

Next let us come to instruction decode; now we have seen that the instruction format is very regular; the opcode is always in the first 6 bits; bit number, 26 to 31. The first source operand; so wherever is there will be in bit number 21 to 25 and the second source operand in bit number 16 to 20; 16 bit immediate data 0 to 15 and 26 immediate data for jump instruction we are not using this; so, ignore this for the time being. So, decoding and register perfecting R S and R T will go on in parallel. Similarly, immediate data will be sign extended; let us see what happens.

(Refer Slide Time: 08:33)

```
ID: A <- Reg [rs];
     B <- Reg [rt];
     Imm <- (IR15)16 ## IR15.0 // sign extend 16-bit immediate field
     Imm1 <- (IR25)6 ## IR25.0 // sign extend 26-bit immediate field
```

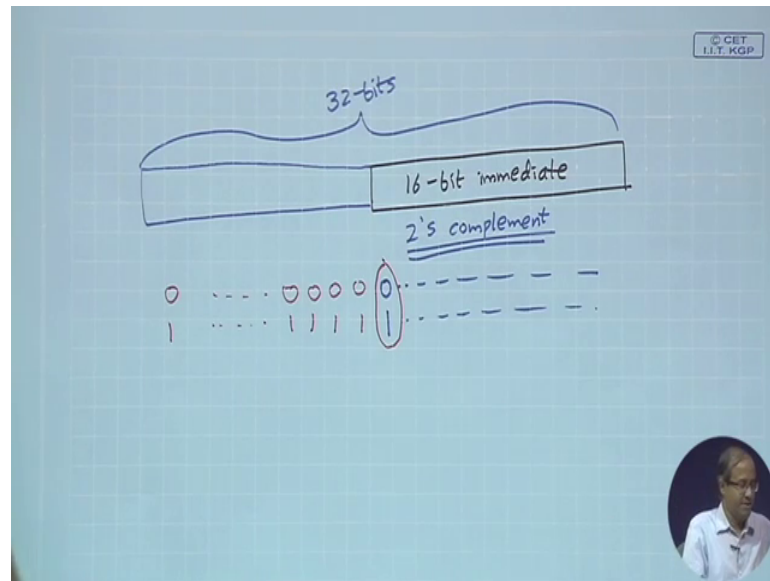
A, B, Imm, Imm1 are temporary registers.

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

These are the micro operations; we are doing register prefetch; you see the two fields are rs and rt. So, we are writing it like this Reg rs; Reg rt; we are fetching the register there are two read ports. So, in the earlier examples of register bank we saw; you recall we assume that there are two read ports and one write port, now you can correlate why we need that.

You see here; we are reading two things at the same time in the same cycle. So, two register are fetched; one you store in A, other you store in B. And this is how sign extension is carried out; the last one you can ignore because this we are not implementing; let us try to understand what we are doing here.

(Refer Slide Time: 09:36)



If you see we have a 16 bit immediate field; we have to extend it to 32 bit. So, what we are doing is as follows; so, we have a 16 bit immediate field. Now, this 16 bit immediate field is considered to be a signed number in 2's complement; this is how it is regarded 2's complement signed number.

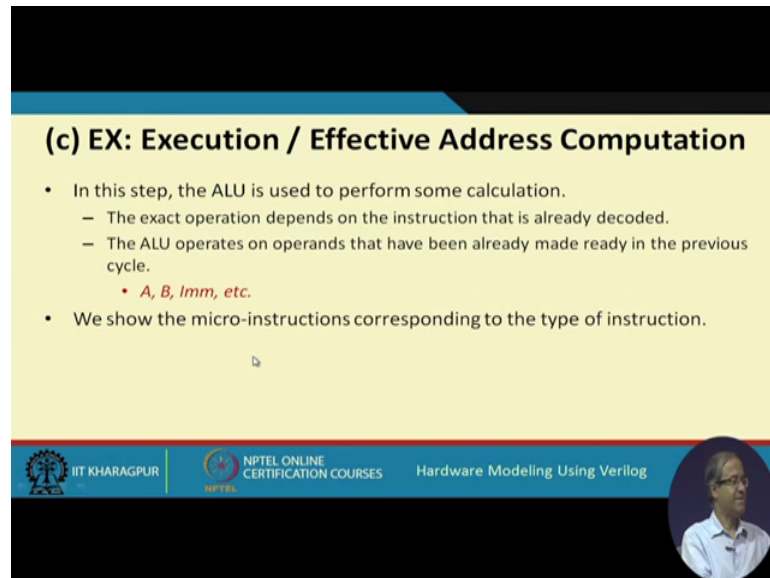
So, if the number is positive; it will start with 0; 0 then anything. If the number is negative; it starts with one then anything. Now, sign extension means we want to extend this number to 32 bits; to make it 32 bits. Now in 2's complement, the rule for sign extension is very easy; it says whatever is the sign bit, you replicate that. If the number is positive just add 16; 0's in the beginning, that will make same number in 32 bits. For negative numbers similarly if the sign is 1; you replicate this first 16 bits with 1; the value remains the same.

So, sign extension means just this you replicate the sign bit 16 times. Now, this we are representing it like this notationally; you see I R; the last sixteen bits I am writing as 0 to 15 like this and this fifteenth bit; I R 15 is a sign bit. This we are replicating it 16 times to the power 16 means; it is just a notation, 16 times replication and this double hash means concatenation.

Well of course, when you write in verilog; we have some other ways of representing, there those curly brackets, concatenation operation, replication operation those are there.

But this is a symbolic notation not in verilog; I am just writing it in symbolic notation and this A, B, I, M etcetera; these are all temporary registers which will be required later.


(Refer Slide Time: 11:56)



(c) EX: Execution / Effective Address Computation

- In this step, the ALU is used to perform some calculation.
 - The exact operation depends on the instruction that is already decoded.
 - The ALU operates on operands that have been already made ready in the previous cycle.
 - *A, B, Imm, etc.*
- We show the micro-instructions corresponding to the type of instruction.

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



Then comes the executions step EX; here we either execute the arithmetic and logic operation whatever is asked for or we calculate the effective address for load and store instructions. Basically in this step the ALU is carrying out some operation; now exact operation will of course, depend on the instruction and the operation will be carried out on numbers which are already generated by the previous stage A, B, I, M, M. So, it will be operating on these numbers A, B, I, M, M etcetera.

(Refer Slide Time: 12:45)

Memory Reference: ALUOut $\leftarrow A + \text{Imm};$	Example: LW R3, 100(R8)
Register-Register ALU Instruction: ALUOut $\leftarrow A \text{ func } B;$	Example: SUB R2, R5, R12
Register-Immediate ALU Instruction: ALUOut $\leftarrow A \text{ func } \text{Imm};$	Example: SUBI R2, R5, 524
Branch: ALUOut $\leftarrow \text{NPC} + \text{Imm};$ cond $\leftarrow (A \text{ op } 0);$	Example: BEQZ R2, Label [op is ==]

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

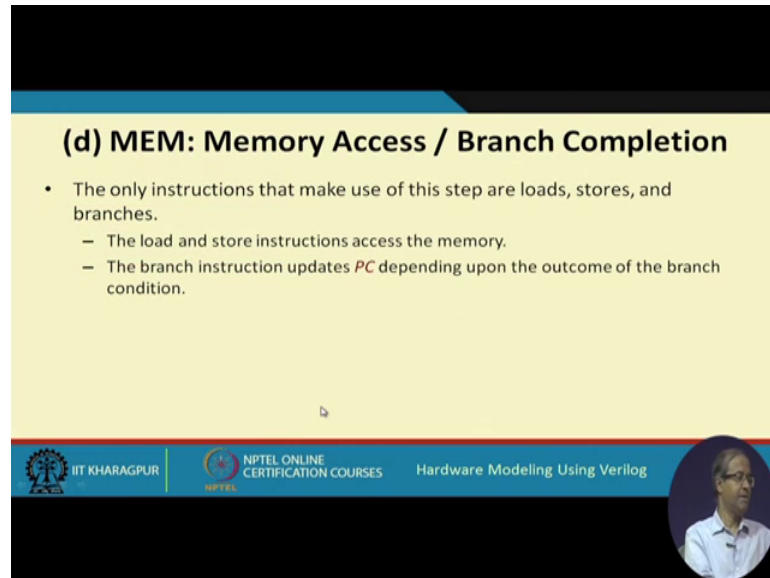
Let us see depending on the type of the instruction; how the steps in the EX stage can vary. Well if it is a memory reference I will load and store instructions; so, I have shown an example of a load. Here we calculate the effective address in this step you see A; will contain the value of R; contain the value of R 8. There is a source register and I M M will consist of this immediate data.

So, a plus I M M; this R 8 plus 100 is calculated and the effective address is temporarily stored in a register called ALU out. Similarly, for register, register ALU operation like this let us say subtract R 2, R 5, R 12; here A is R 5 and B is R 12 and func is actually the kind of the function. Subtract means this is subtraction; so, you actually carry out the operation specified by the instruction on the operands which will be A and B and again the store the result and the result is stored in ALU out. And if it is an immediate operand there is immediate operand; register immediate, then it is similar to this because second operation is not B; second operand will be I M M; R 5 will be A and 524 is I M M.

And for branch you do these things, here the level of whatever is the offset that will be the last 16 bits; that will be your I M M; that will be added to NPC. Because you still now do not know whether you are taking the branch or not taking the branch. But in case you take the branch, where to go you are making that ready, you are adding the program counter that NPC with this offset and getting ready with the new address in case you decide later that well I have to take a branch.

And also you calculate the condition because there is branch equal to 0; you check whether A; R 2 is a equal to 0 or not and that result of this comparison you store in a flip flop; a 1 bit register call cond, this will be required or used in the next stage.

(Refer Slide Time: 15:06)



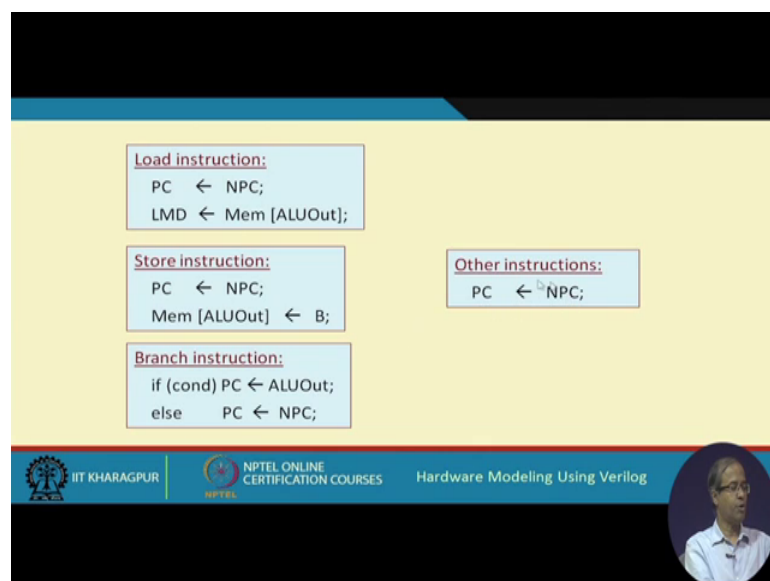
(d) MEM: Memory Access / Branch Completion

- The only instructions that make use of this step are loads, stores, and branches.
 - The load and store instructions access the memory.
 - The branch instruction updates *PC* depending upon the outcome of the branch condition.

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

Now, come to the MEM: MEM here we do memory access or branch completion. So, only load store and branches they will use this stage; other instructions they do not do anything in MEM; let us see what is be done.

(Refer Slide Time: 15:23)



Load instruction:
PC ← NPC;
LMD ← Mem [ALUOut];

Store instruction:
PC ← NPC;
Mem [ALUOut] ← B;

Branch instruction:
if (cond) PC ← ALUOut;
else PC ← NPC;

Other instructions:
PC ← NPC;

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

For the load instruction; you simply do this because load instruction the PC will be the NPC. So, just the NPC; you load into PC and you do a memory read because in the previous stage already the address of the operand is stored in ALU out. So, you access memory with that address and the data that is coming out; you temporarily store in an register LMD; load memory data, this is also temporary register.

For store instruction; it is just the reverse well first thing is again NPC goes to PC and B whatever is there is in B; that you store in MEM; ALU out; this is what you do because for a store instruction your B is the target; second operand is the target; that will be stored here ALU out and for branch instruction; in the earliest stage we have already computed cond.

So, now you check if cond is 1; then there is a branch. So, the new value of address you have calculated that goes to PC or otherwise this NPC just the next instruction plus 1; that goes to PC; this is what is done during the MEM stage.

(Refer Slide Time: 17:00)

(e) WB: Register Write Back

- In this step, the result is written back into the register file.
 - Result may come from the ALU.
 - Result may come from the memory system (viz. a LOAD instruction).
- The position of the destination register in the instruction word depends on the instruction → *already known after decoding has been done.*

	31	26	25	21	20	16	15	11	10	6	5	0
R-type	opcode	rs	rt	rd	shamt	funct						

	31	26	25	21	20	16	15	0			
I-type	opcode	rs	rt	Immediate Data							

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES

Hardware Modeling Using Verilog

And for all other instruction; just you copy NPC to PC. Lastly in the W B stage; you have to write back the register to store the result. Result is written back to the register, now the result that that you are writing back; it can either come from the memory like a subtract, add, multiply instruction or it can be load instruction now the data is coming from memory. Because you see earlier the data from the memory is coming into the register for LMD; so, from LMD; it will go.

Now, one thing you see for a R type instruction; your destination is R D from bit number 11 to 15 and I type instruction; your destination is R T bit number 16 to 20. So, depending on the type of the instruction, you will have to decide on the destination register.

(Refer Slide Time: 18:00)

The slide displays three instruction execution examples in light blue boxes on a yellow background:

- Register-Register ALU Instruction:**
Reg [rd] ← ALUOut;
- Register-Immediate ALU Instruction:**
Reg [rt] ← ALUOut;
- Load Instruction:**
Reg [rt] ← LMD;

The footer contains the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and the text "Hardware Modeling Using Verilog". A circular inset shows a speaker.

So, let us see; if it is a register; register ALU instruction; that means, this one, first one. So, R D will be your register target; so, ALU out will be stored in Reg rd, but if it is a register immediate; it will be stored in Reg rt because rt is the target.

(Refer Slide Time: 18:32)

The slide features the title "SOME EXAMPLE INSTRUCTION EXECUTION" in bold black text centered on a yellow background.

The footer contains the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and the text "Hardware Modeling Using Verilog". A circular inset shows a speaker.

Similarly, for a load instruction; the data is stored in LMD temporarily that data will be loaded into Reg rt. So, this you have to remember use rd here; and rt; rt here. Now, let us look at sum instruction execution in completion; the complete set of micro operations. So, if you look at this you will understand what is happening.

(Refer Slide Time: 18:38)

ADD R2, R5, R10		ADDI R2, R5, 150	
IF	IR ← Mem [PC]; NPC ← PC + 1 ;	IF	IR ← Mem [PC]; NPC ← PC + 1 ;
ID	A ← Reg [rs]; B ← Reg [rt];	ID	A ← Reg [rs]; Imm ← (IR ₁₅) ¹⁶ ## IR _{15,0}
EX	ALUOut ← A + B;	EX	ALUOut ← A + Imm;
MEM	PC ← NPC;	MEM	PC ← NPC;
WB	Reg [rd] ← ALUOut;	WB	Reg [rt] ← ALUOut;

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

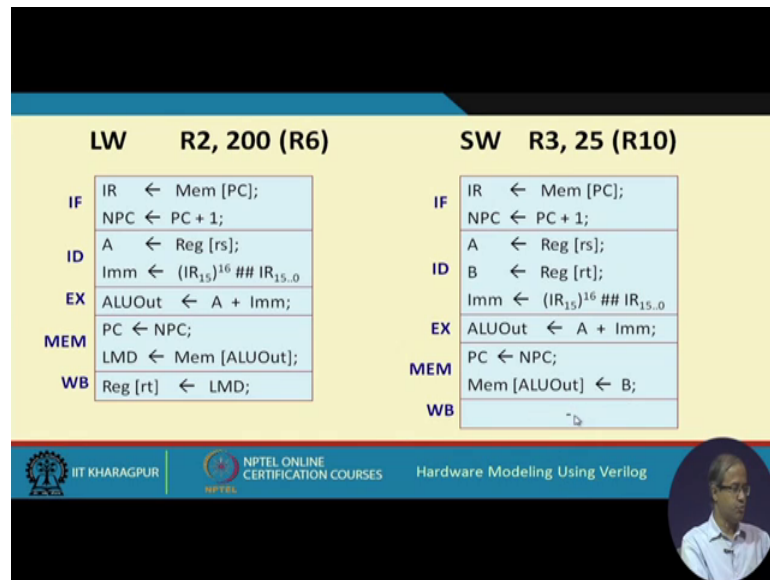
Take this instruction add R 2, R 5, R 10. So, R 5 and R 10 is added result is stored in R 2; let us see in the five stages what is happening; for IF you are fetching this instruction in I R and incrementing PC that goes to NPC; second stage well I M M; I am not showing because this instruction does not require I M M; it only requires reg rs and reg rt. So, they are stored in A and B; so, in the EX stage; it will decode it is an add. So, there will be an addition operation a plus b will be carried out the result will be stored in ALU out.

MEM nothing other than NPC goes to PC this copying nothing else is done and in the right back; whatever ALU out is computed that will finally be written to the register rd; rd is R 2. So, in this way this instruction gets executed, these are the steps. Let us take an another example; this is an immediate, add immediate instruction; there is an immediate operand here.

So, instruction fetch is identical; so, fetch an instruction increment the PC; here there is only one source operand R5. So, I am not showing B; I am only showing A; the relevant ones I am only showing; A equal to Reg rs; this is rs and immediate data is 150. So, in I M M; this 150 gets sign extended you store in I M M.

In EX you add A and IMM; so, R5 and 150 gets added MEM; same NPC goes to PC and WB. Similar, but instead of rd; here it goes to rt, because in this instruction format only two registers are specified rs and rt.

(Refer Slide Time: 20:42)



Let us look at a load instruction load R to 200; R 6. So, here this 200 will be added to R 6 to get the memory address from where data will be loaded into rt. So, IF is again same as the earlier instructions this ID is also same as in add immediate; the first operand R 6 is loaded in a and the immediate data 200 is loaded and sign extended IMM. In EX for load and store instruction; this will always be add. So, this 200; R 6 are added result in ALU out.

And in MEM well other than NPC 2 PC; another thing is actually done; that means, memory access. This ALU out actually contains the memory address; MEM of that; that is read and the data get stored in LMD and lastly in WB that L m D is written into register rt; where rt is R 2.



Now, lastly let us look at a store instruction; it will be quite similar, but in store instruction you see in the last step there is nothing to do. First time IF is again identical well in ID well here you are actually loading rs and rt both; R 3 and R 10; it depends on the implementation; this is just one implementation I am showing. Now in our actually implementation, we will make it a little different.

So, let us say the first one is R 3 is rs; R 10 is rt; let us say that is the convention I have shown here in this. This ALU out a plus and same way we compute the address and here you write; B contains this R 3. So, B will actually get stored in R 10 is A, R 3 is B. So, so when you say store; this B will get stored into the memory and this store operation is complete within MEM. So, in W B you do not have anything to do now this is a gap.

(Refer Slide Time: 23:09)

BEQZ R3, Label

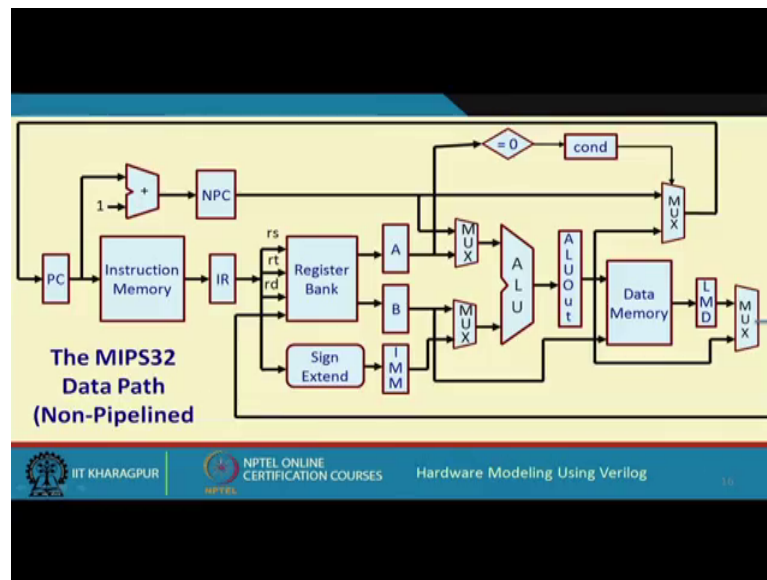
IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 1;
ID	A \leftarrow Reg [rs]; Imm \leftarrow (IR ₁₅) ¹⁶ ## IR _{15,0}
EX	ALUOut \leftarrow NPC + Imm; cond \leftarrow (A == 0);
MEM	PC \leftarrow NPC; if (cond) PC \leftarrow ALUOut;
WB	↳ -



 NPTEL ONLINE CERTIFICATION COURSES
 Hardware Modeling Using Verilog

And one example of a branch instruction; branch equal to 0; it checks whether R 3 is 0 or not; if it is 0 then it jumps to level. You say what is done here instruction fetch is again same; instruction decode R 3 is fetched into A and immediate whatever offset is there that is sign extended in I M M; in EX stage we are adding R 3 with this; not R 3 it is sorry, we are just adding the program counter the NPC with the offset to compute the branch address in case you decide to branch later. So, you are adding this two up.

And also you are checking the branch condition because the branch condition was equal to 0; you check this R 3 is already loaded in A, you check whether A is equal to equal to 0 or not; just a comparator. In the result of the comparison you store in this flip flop 1 bit register cond. In MEM, you check this cond NPC; p c is of course, the first thing that you do, but you may have to override it; if cond your ALU out goes to PC. So, here you can also write it like this; if cond PC equal to ALU out else PC equal to NPC you can also write like that; W B again there is nothing for branch.

(Refer Slide Time: 24:41)



Now, you see whatever we have talked about just over all the data path can be very conveniently represented or drawn like this. This is our non pipelined data path of the MIPS 32; you can identify the different stages, you see here this is the instruction fetch; where this is the memory which stores the instruction, PC is the address and you read the memory. The instruction is read out you store it in the instruction register I R and there is also an operation NPC equal to PC plus 1. So, there is an adder that adds 1 to PC and result stores into NPC; so, this is your instructions fetch.

Now, instruction decode of course, the decoder is not shown. So, in the I R that opcode part of I R is decoded and the rs and rt part; the two source registers they are pre fetched; one of them is loaded in R A; another is loaded in B and the last 16 bit offset; that is sign extended, that is stored in I M M.

In the EX stage; you operate on two numbers well of course, the function will depend on the instruction type; I am not showing that whether it is add, subtract, multiply what. But the data that you are operating on that can be different, the first data can either be A or NPC; NPC will be for the branch instruction recall because NPC will be added to something. So, it can be either NPC or it will be A.

The second multiplex will be selecting B or the immediate data; this also depends on the instruction and the ALU stores the result in ALU out. And in the EX stage I said; also you have a comparator which checks whether A equal to 0 or not and the result you store

in a 1 bit register cond. In the MEM stage; you actually do either a read or write. So, the memory address is obtained from ALU out and in case you want to do a write; the data is in B; B is written here.

And in case you do read; the data read will go to LMD and this is the write back stage; W B well see W B is nothing extra just as a multiplexer which writes back something into the register bank because register bank is already here; there is a third write port using rd. So, rd is already known that what was the destination register and the data can be either coming from LMD for load instruction or from ALU instruction it will come from ALU out. That data will be stored in register; so, you see this is the overall architectural execution process of the MIPS.

Now, because the instruction set was so simple because the instruction encoding and decoding is so, simple. You see the whole data path I can fit in a single slide, but you think of a more complex processor; is it possible to do this? No you cannot, you can never do this for a more complex instruction set.

So, for the risk or reduced instruction set computer this is the main advantage. Implementation becomes very very simple; so, this architecture which I have shown. So, for the data path; for a non pipeline case; in the next lecture we shall be showing how this can be partitioned into well defined pipelines, what will be the stages and how we have to modify the micro operations to suit the pipelining requirements. So, with this we come to the end of this lecture; we shall be looking at how to extend this data path to a pipeline data path in our next lecture.

Thank you.