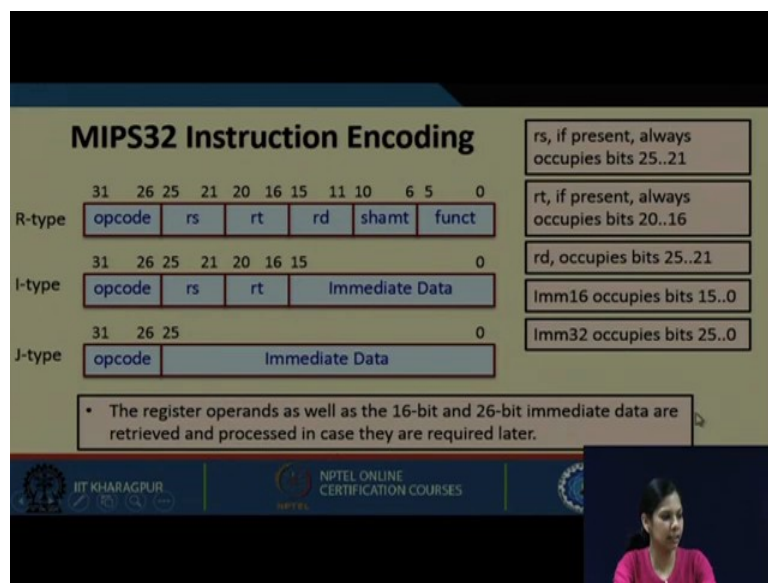


Computer Architecture and Organization
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 21
MIPS Implementation (Part 1)

Welcome to the next lecture. Till now what we have seen that how we can design a control unit. We have also seen both hardware and microprogrammed control unit design methods. So, in this lecture we will be now looking into MIPS implementation. We have seen the general way of designing a control unit. We will be in this lecture specifically seeing how in MIPS the data path is there, how the instructions are actually getting executed with respect to MIPS instruction set.

(Refer Slide Time: 01:14)

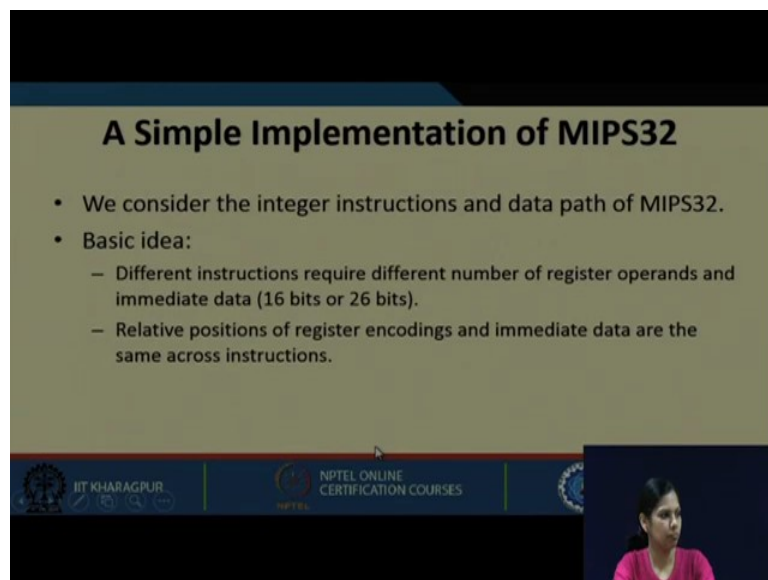


We have already discussed this in week 2 lectures, MIPS32 instruction encoding. So, we have R type instruction, we have I type instruction and we have J type instruction. In R type instruction, we have an opcode field, we have 3 register fields (2 source register, one destination register), we have shift amount value, and this is opcode extension function. In the I type that is immediate type instructions, we have opcode, 2 registers, and we have an 16 bit immediate value.

Similarly, in J type instructions, we have an opcode, and we have a immediate value of 26 bit. So, rs if present always occupies bits 21-25. If rt is present it always occupies bits 16-20. Similarly, rd occupies from bits 11-15.

This immediate field contains bits 0-15; it is 16 bit. This immediate field occupies 0-25; it can be extended by 2 more bits, later we can see. So, the register operands as well as 16 bit and 26 bit immediate operand are retrieved and processed in case they are required later. So, as you see that this is the opcode. These are source register and destination register, these are the immediate data. So, what we can do we can retrieve these data in advance.

(Refer Slide Time: 03:48)



A Simple Implementation of MIPS32

- We consider the integer instructions and data path of MIPS32.
- Basic idea:
 - Different instructions require different number of register operands and immediate data (16 bits or 26 bits).
 - Relative positions of register encodings and immediate data are the same across instructions.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, in a simple implementation of MIPS, we consider the integer instructions and data path of MIPS. So, what the basic idea goes here is that different instructions require different number of register operands. And relative positions of the register encoding and immediate data are the same across instructions. So, by this what we mean that we use any instruction, does not matter it can be a J type it can be I type. But this is fixed that from this particular bit to this particular bit, it will have this particular data; from this particular bit to this particular bit, it will be this particular register; from this to this it will be another register. So, that is fixed. So, this information is known to us.

(Refer Slide Time: 04:43)

The slide contains the following text:

- A Naïve Approach:
 - After fetching and decoding an instruction, identify the exact register(s) and/or immediate operands to use, and handle them accordingly.
 - The number of register fetches and immediate operand processing will vary from instruction to instruction.
 - We do not utilize the possible overlapping of operations to make instruction execution faster.
 - Before instruction decoding is complete, fetch the register operands and immediate data in case they are required later.

At the bottom of the slide, there are three logos: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA.

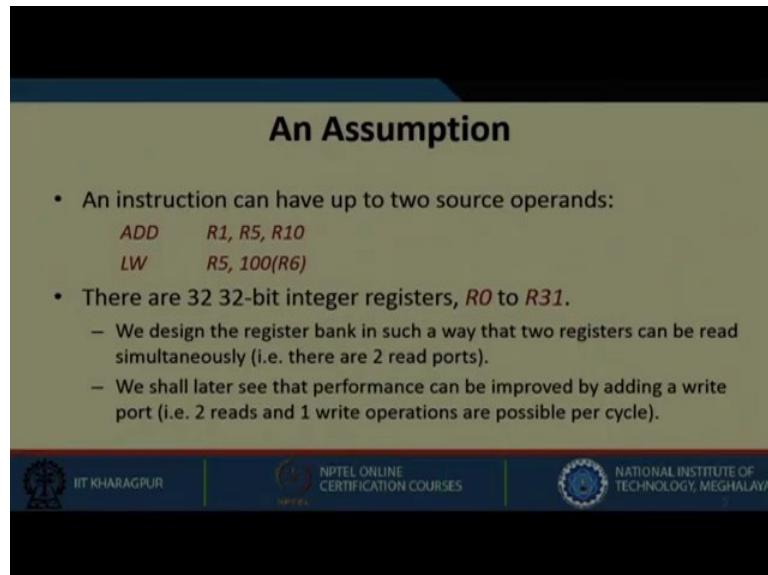
Let us take a naive approach. In a naive approach what happens after fetching and decoding an instruction, we identify the exact register or immediate operand to use and handle them accordingly. What we are saying that we will fetch an instruction, we will decode that instruction, and then we will come to know that this particular instruction performs this particular task. So, we can extract the register, if it is for an immediate operand we can extract the immediate operand. So, we are not doing something well in advance rather after decoding it we are starting to do all these things.

The number of register fetches and immediate operand processing will vary from instruction to instruction; obviously. We do not utilize the possible overlapping of operations to make the instruction execution faster. If we are just fetching one by one by one, then we really cannot take the advantage of this overlapped execution of instruction, that is pipeline. So, we will not be able to take advantage of that. Why because, we are fetching these instruction one by one, and then we are decoding then we are getting all the other immediate value or register value, etc.

So, before instruction decoding is complete, we fetch the register operands and immediate data in case they are required later. So, this is a better way. Before the decoding is performed, we want to fetch --- we already know that this particular bit will be a register, this particular bit will be an immediate data, this will be destination register, this will be source register. So, why not let us take those, fetch and keep in some proper

place. If it is not required later we will not use it, but at least if it is required, then it will be very easy for us to get the data; we do not have to fetch it again.

(Refer Slide Time: 07:08)



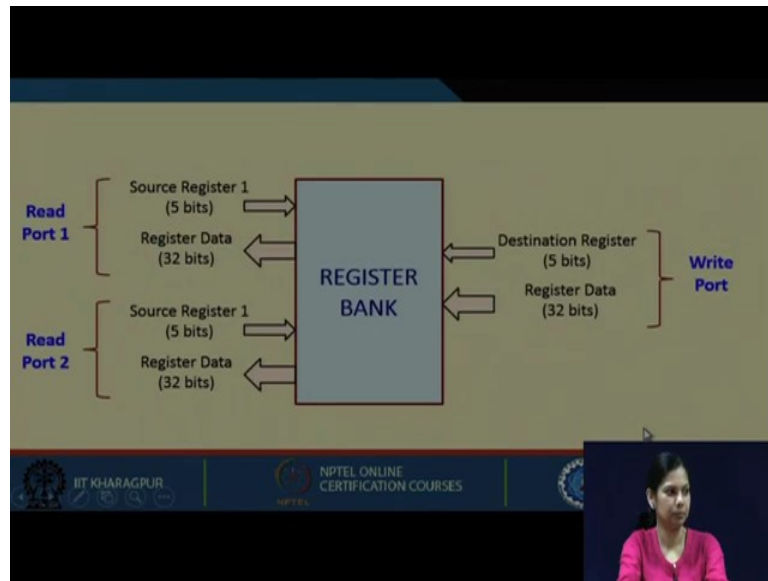
An Assumption

- An instruction can have up to two source operands:
ADD R1, R5, R10
LW R5, 100(R6)
- There are 32 32-bit integer registers, *R0* to *R31*.
 - We design the register bank in such a way that two registers can be read simultaneously (i.e. there are 2 read ports).
 - We shall later see that performance can be improved by adding a write port (i.e. 2 reads and 1 write operations are possible per cycle).

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

So, this is an assumption that an instruction can have up to 2 source operands. Basically one is `ADD R1,R5,R10` and for `LW R5,(100)R6`. So, there are 32, 32-bit registers, `R0` to `R31`. We design the register bank in such a way that 2 registers can be read simultaneously. That is, there are 2 read ports we already have seen in multibus architecture, that this might be possible that a particular register has two read ports and one write port. We shall see later that the performance can be improved by adding a write port; that is, 2 read and 1 write are possible per cycle.

(Refer Slide Time: 08:10)



So, this is the story. All together we have a register bank where we can read from 2 registers and we can write into 1 register. So, read port 1, read port 2, and we have one write port. So, source register 1 will be 5 bits, and the data will be 32-bit, source register 2 will be 5-bits and the register data can be 32-bit, and this is the destination register.

(Refer Slide Time: 08:42)

- A Speculative Approach:
 - Here we try to eliminate the time required to fetch the register operands and process the immediate data.
 - When an instruction is decoded, at the same time we fetch the register operands and also process the immediate data (i.e. sign extend).
 - Possible because their locations in the instruction word are fixed.
 - If the operands are required, they are already available (no extra time required).
 - If the operands are not required, they are ignored.

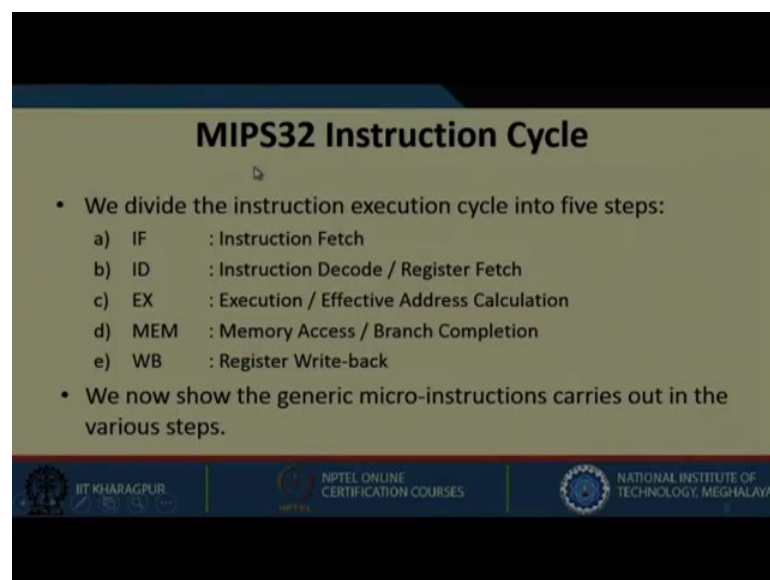
The slide contains the text of the bullet points above. It also features the same logos and video inset as the previous slide.

Let us now come to a speculative approach. Let us try to speculate something here; we try to eliminate the time required to fetch the register operands and process the immediate data. So, as we said that when an instruction is decoded at the same time we

fetch the register operands and also process the immediate data; that means, we have already seen that in MIPS architecture the immediate data is sign extended to make it 32 bit.

So, all these things can be done once it is decoded. We really do not know at this point of time whether it is it will required or not. But still let us do it, because their locations in instruction word are fixed, and because of this fixed location we are able to do this. If the operands are required, they are already available; no extra time will be required because we have already done this fetching; and if the operands are not required, they are simply ignored.

(Refer Slide Time: 10:30)



The slide is titled "MIPS32 Instruction Cycle" and lists five steps of the instruction execution cycle. The steps are: a) IF: Instruction Fetch, b) ID: Instruction Decode / Register Fetch, c) EX: Execution / Effective Address Calculation, d) MEM: Memory Access / Branch Completion, and e) WB: Register Write-back. The slide also includes a footer with logos for IIT Kharagpur, NPTEL Online Certification Courses, and National Institute of Technology, Meghalaya.

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 | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, MIPS32 instruction cycle is divided into certain steps. So, what are the steps? Instruction fetch, instruction decode or we can say register fetch, execute where effective address calculation is also done, this is memory access and branch completion, and write back to a register. We now show the generic micro instructions that are carried out in the various steps.

(Refer Slide Time: 11:14)

(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 (i.e. 4 bytes).
 - 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*.

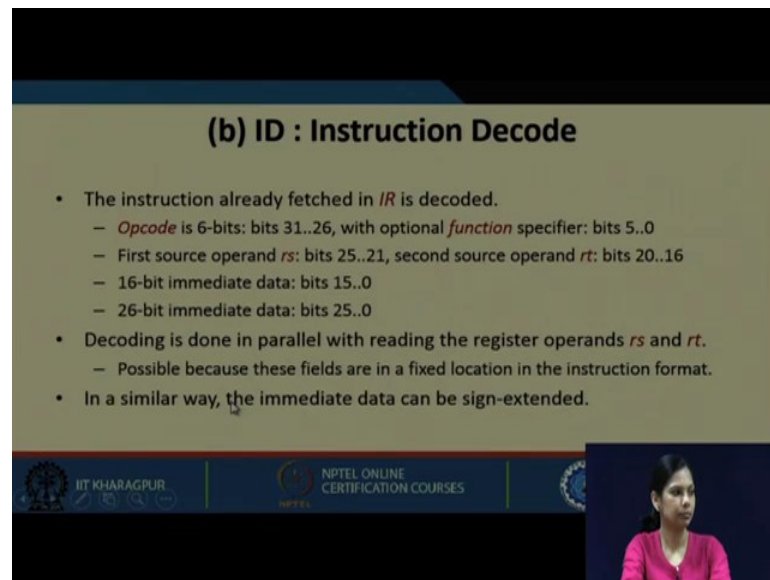
```
IF:  IR ← Mem [PC];  
     NPC ← PC + 4;
```

The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES. A small video inset of a woman is visible in the bottom right corner.

In the instruction fetch what happens, we know we fetch an instruction. As we have already seen for single bus architecture. Here also it is pretty same, but it is specific to MIPS. Here the instruction pointed to by PC is fetched from memory, and also the next value of PC is computed. So, every MIPS instruction is 32 bits, that is 4 bytes. For a branch instruction, new value of the PC may be the target address. So, PC is not updated in this stage; the new value is stored in a register called NPC.

So, this is little bit different than we have done earlier. What we are doing we are updating the PC value, and if there is a branch later at that point, we are doing Yin at certain stage, and that Yin can be added with that particular offset to go to the branch location. There we were doing like this. But in MIPS, we are updating the PC value, but not updating it in the PC register. They are adding the PC value, and the new value is stored in another register. So, for this purpose they have kept another register called NPC, where the updated PC value is stored and not in the PC. So, we do Mem[PC]. So, the content of memory location pointed by PC is brought into IR, and PC is incremented by 4 and it is stored in NPC.

(Refer Slide Time: 13:16)



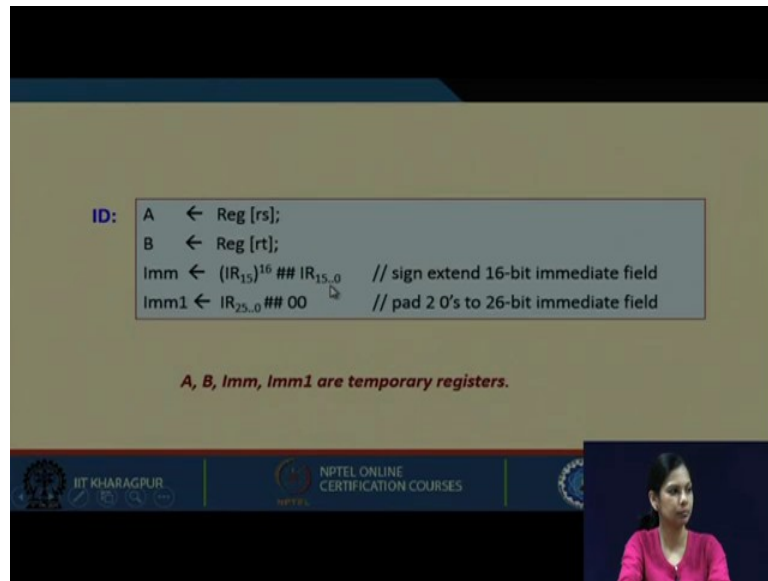
(b) ID : Instruction Decode

- The instruction already fetched in *IR* is decoded.
 - *Opcode* is 6-bits: bits 31..26, with optional *function* specifier: bits 5..0
 - 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 can be sign-extended.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see what happens in instruction decode. The instruction already fetched in IR is now decoded. As we said that the opcode is a 6-bit from 0 to 5. These are also stored; first the source operand *rs*, second the source operand *rt*, 16-bit immediate data and 26-bit immediate data are also fetched. So, all these are fetched we do not know whether we will be requiring it or not, but we have fetched it decoding is done in parallel with reading the register operand *rs* and *rt*. And this is done within the processor, because our instruction is in IR, and from IR we are taking it one by one. Possible, because these fields are in fixed location in the instruction format. In a similar way the immediate data can be sign extended. So, the immediate data can be sign extended to make it 32-bit.

(Refer Slide Time: 14:29)



The slide displays the following register assignments:

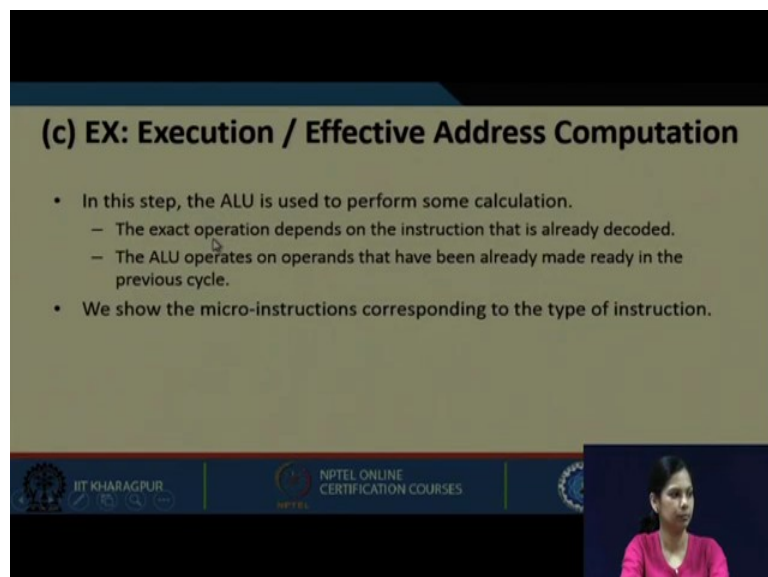
```
ID: A ← Reg [rs];
     B ← Reg [rt];
     Imm ← (IR15)16 ## IR15:0 // sign extend 16-bit immediate field
     Imm1 ← IR25:0 ## 00 // pad 2 0's to 26-bit immediate field
```

A, B, Imm, Imm1 are temporary registers.

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video inset of a presenter.

So, this is what we are doing. In A we are bringing Reg[rs], in B we are bringing Reg[rt]. Immediate data which is 0 to 15 we are sign extending with the first bit, that is IR₁₅ 16 times. And the next immediate field is padded with 2 zeros. So, this 26 bits are kept. So, Imm and Imm1 are temporary registers that are loaded in the instruction decoding phase with these particular values.

(Refer Slide Time: 15:29)



(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.
- We show the micro-instructions corresponding to the type of instruction.

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video inset of a presenter.

Let us see what happens in execution phase. In the execution phase, the effective address computation is performed. So, 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 cycles. We have already fetched and kept it in A and B registers in the decoding phase, and it is an ALU operation; then finally, in the execute phase the operation specified can be performed. So, we show the micro operations corresponding to the type of instruction.

(Refer Slide Time: 16:25)

The slide displays four categories of ALU operations with their respective formulas and examples:

- Memory Reference:**
ALUOut \leftarrow A + Imm;
Example: LW R3, 100(R8)
- Register-Register ALU Instruction:**
ALUOut \leftarrow A func B;
Example: SUB R2, R5, R12
[operation specified by func field (bits 5..0)]
- Register-Immediate ALU Instruction:**
ALUOut \leftarrow A func Imm;
Example: SUBI R2, R5, 524
[operation specified by func field (bits 5..0)]
- Branch:**
ALUOut \leftarrow NPC + (Imm \ll 2);
cond \leftarrow (A op 0);
Example: BEQZ R2, Label
[op is ==]

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter in the bottom right corner.

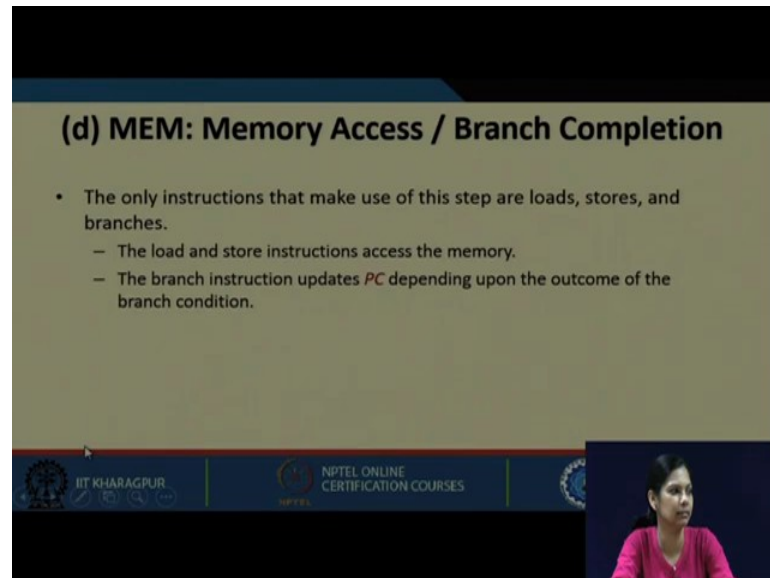
Now, in execute phase what can happen? If it is a LW then this is an immediate value added with R8. So, R8 goes in A, added with the immediate value that is 100, which goes in the output of ALU. For register to register, both A and B are present for this instruction; R5 and R12 both are present, it will be added or subtracted depending on the function and it will be stored in ALUOut.

Similarly, register immediate. In this case, R5 is subtracted with an immediate value. So, R5 is available in A, and this immediate value is available in Imm, and this function that is subtraction can be performed and the output is available in ALUOut.

Now, for branch, what happens? For the branch the immediate value that we have got it needs to be left shifted twice, and then added with NPC because NPC contains the incremented value that will come to ALUOut. And if we have a branch instruction like BEQZ; that means, if R2 equal to 0 then only branch. We have to check for this condition, and this condition will be an operation with 0. So, if this particular operation if

this condition is satisfied then you will branch. So, it will do some operation and it will set the condition and accordingly branch will take place.

(Refer Slide Time: 18:33)



(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

Now, what happens in MEM --- memory access or branch completion? The only instructions that make use of this step are load and store, and of course branches. The load-store instruction accesses the memory. So, the memory operation will actually happen here. The branch instruction updates PC depending upon the outcome of the branch condition. So, this also happens here. So, these are the two things that happen in MEM phase.

(Refer Slide Time: 19:06)

The slide displays the execution logic for different instruction types:

- Load instruction:**
PC \leftarrow NPC;
LMD \leftarrow Mem [ALUOut];
- Store instruction:**
PC \leftarrow NPC;
Mem [ALUOut] \leftarrow B;
- Branch instruction:**
if (cond) PC \leftarrow ALUOut;
else PC \leftarrow NPC;
- Other instructions:**
PC \leftarrow NPC;

Logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA are visible at the bottom.

So, now NPC will be loaded in PC, and for the load instruction output of ALU location pointed by that will come to LMD. Similarly, NPC will come to PC, and then B will be put into that particular location whose address is in ALUOut; and for the branch what happens if the condition is satisfied, then that ALUOut will go to PC, and else NPC, which we have already calculated will be loaded to PC. And for all other instruction, we have already calculated the PC value in NPC, which will be put in PC.

(Refer Slide Time: 20:25)

(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 \rightarrow *already known after decoding has been done.*

Instruction formats are shown below:

R-type

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

I-type

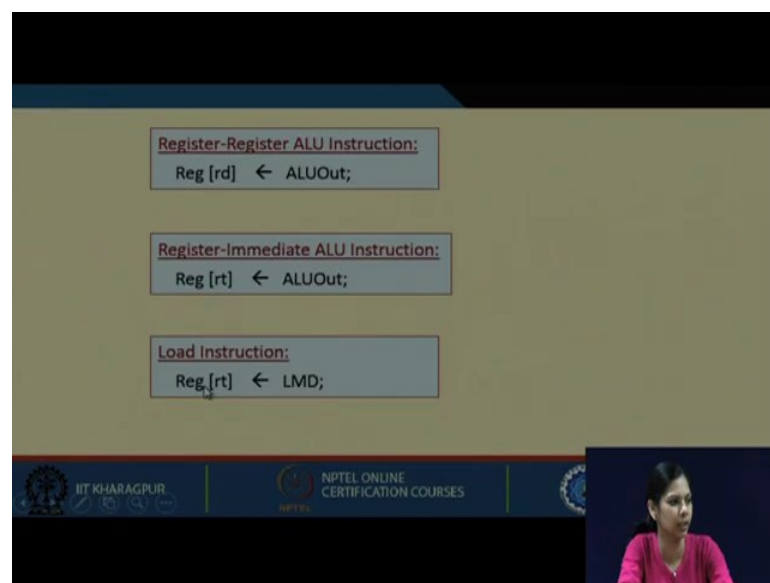
31	26	25	21	20	16	15	0				
opcode		rs	rt	Immediate Data							

Logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA are visible at the bottom.

Let us see in Write Back what happens; register write back occurs in this step. The result is written back into the register file. Result may come from memory system via load as well. The position of the destination register in the instruction word depends on the instruction already known in the decoding phase. So, this is basically the destination register in this particular case, and this is the destination register.

So, the position of the destination register we already know from the decoding phase, and then the result may be put in there in this particular step.

(Refer Slide Time: 21:16)




So, whatever value was there in ALU can be put in for register transfer. Here also ALUOut will be put in Reg[rt]; and in load instruction in LMD we have stored that value, now it will go to the required register.

Let us see some example instructions. Now we have seen step by step how in MIPS the instructions are executed.

(Refer Slide Time: 22:09)

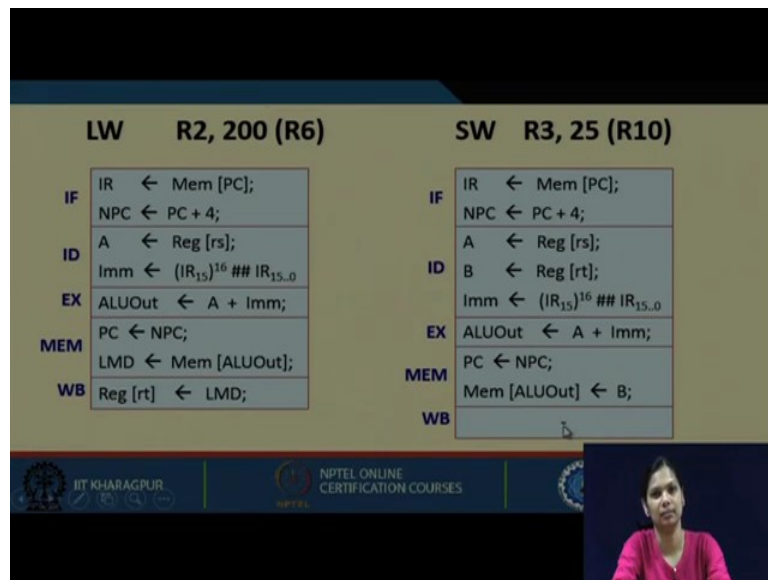
	ADD R2, R5, R10	ADDI R2, R5, 150
IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 4;	IF IR \leftarrow Mem [PC]; NPC \leftarrow PC + 4;
ID	A \leftarrow Reg [rs]; B \leftarrow Reg [rt];	ID A \leftarrow Reg [rs]; Imm \leftarrow (IR ₁₅) ¹⁶ ## IR _{15..0}
EX	ALUOut \leftarrow A + B;	EX ALUOut \leftarrow A + Imm;
MEM	PC \leftarrow NPC;	MEM PC \leftarrow NPC;
WB	Reg [rd] \leftarrow ALUOut;	WB Reg [rt] \leftarrow ALUOut;



Now, let us see a complete execution of an instruction ADD R2,R5,R10. So, in the instruction fetch phase IR will have Mem[PC]. So, this entire instruction is in IR, and NPC will have PC plus 4. Similarly A will have Reg[rs], B will have Reg[rt]. And in the execute phase both the source operands are added, and it is available in ALUOut. And then in the MEM phase the PC is loaded with NPC value, and finally, the value of ALUOut will be put it into the destination register, that is R2 here.

So, in five steps we are executing it, but for all instruction all these steps will be required, let us see how we can add an immediate value. In this case, this is an immediate value. So, this immediate value which is 16-bits concatenated with a MSB value, and we get the total 32-bit immediate value, and A is R5 here. Finally, ALUOut we will be adding this value with immediate value, and NPC is loaded in PC in this particular step, and finally, we write back the output into R2 that is the destination register here.

(Refer Slide Time: 24:01)



Now for load instruction For load instruction, we can see that similarly in instruction fetch these 2 steps will be performed, A will have the source operand, that is R6 here, immediate value will be stored in Imm, we will add R6 with 200 and it will be stored in ALUOut. And then NPC will be put in PC, and memory operation is performed here. So, the output of ALUOut from this particular memory location, we need to read the value, that is what we are doing which is loaded in LMD. Now LMD contains the value that should be put in R2. So, LMD will be stored back here.

Similarly, for storing what we have to do, we fetch and decode. After decoding same way we are adding this and this immediate value, and this register value NPC is loaded to PC. And now instead of getting it from the memory, what we are doing? The value of B that is in R3 is stored in Mem[ALUOut], because ALUOut is the location. We are storing B into Mem[ALUOut], and in Write Back there will have nothing to store.

(Refer Slide Time: 26:10)

BEQZ R3, Label

IF	IR \leftarrow Mem [PC]; NPC \leftarrow PC + 4;
ID	A \leftarrow Reg [rs]; Imm \leftarrow (IR ₁₅) ¹⁶ ## IR _{15,0}
EX	ALUOut \leftarrow NPC + (Imm \ll 2); cond \leftarrow (A == 0);
MEM	PC \leftarrow NPC; if (cond) PC \leftarrow ALUOut;
WB	-

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter in the bottom right corner.

Let us see the next instruction branch if equal to 0. So, what we are doing here. So, in the instruction fetch and in the decode in the same way we are fetching it. In ALUOut what we are doing? We are adding NPC with the immediate value, which is left shifted twice and finally, we are putting that condition based on R3. R3 is loaded in A. If we are checking this condition if R3 equals to 0 or not, according to this cond will be set. If it is 0, then only branch will take place. So, otherwise NPC will be PC. So, if the condition is met then ALUOut will be put in PC, else this NPC will be in PC, and there will be nothing in the Write Back phase. So, this is how the branch instruction is executed in MIPS.

So, we have come to end of lecture 21. In this particular lecture, we have seen that how the instructions are executed in MIPS architecture. How the data part is there in MIPS and how the instructions are executed.

Thank you.