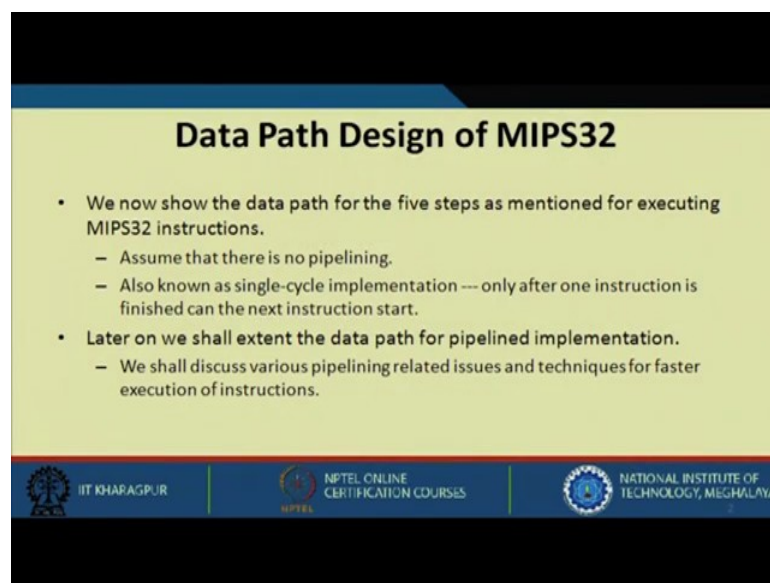


Computer Architecture and Organization
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 22
MIPS Implementation (Part 2)

Welcome to the last lecture of week 4. So, we have already discussed about MIPS implementation. We will continue with that here.

(Refer Slide Time: 00:39)



Data Path Design of MIPS32

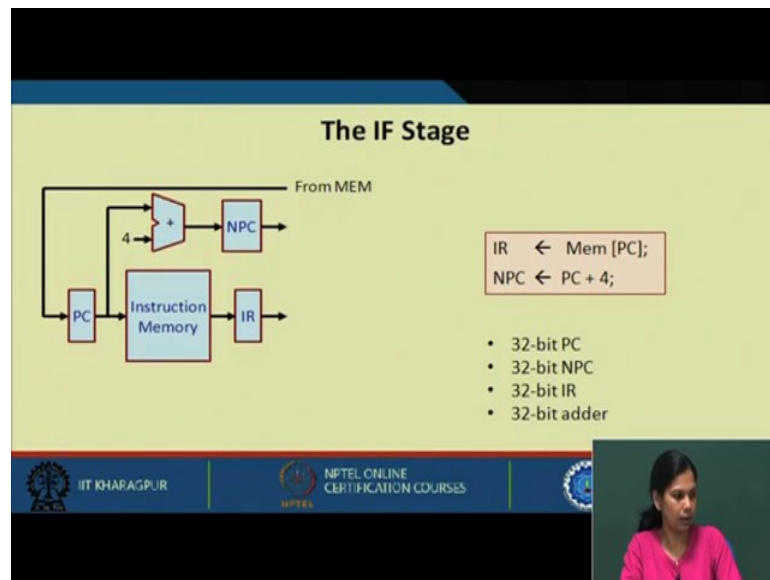
- We now show the data path for the five steps as mentioned for executing MIPS32 instructions.
 - Assume that there is no pipelining.
 - Also known as single-cycle implementation --- only after one instruction is finished can the next instruction start.
- Later on we shall extend the data path for pipelined implementation.
 - We shall discuss various pipelining related issues and techniques for faster execution of instructions.



First, the data path design of MIPS. Individually, we have seen that how we are actually executing a particular instruction, what are the micro-operations that are required to be executed. How the data path is actually designed we will see here.

We now show the data path for the five steps as mentioned for executing MIPS32 instructions, and we are assuming here that there will be no pipelining. This is also known as single-cycle implementation; only after one instruction is finished, can the next instruction start. In a single cycle we execute one instruction fully. And then we move to the next instruction, and again we move to the next instruction after executing the previous one. Later we shall extend the data path for pipeline implementation; we shall discuss various pipelining issues related to the techniques for faster execution of instruction in later phase of this particular course.

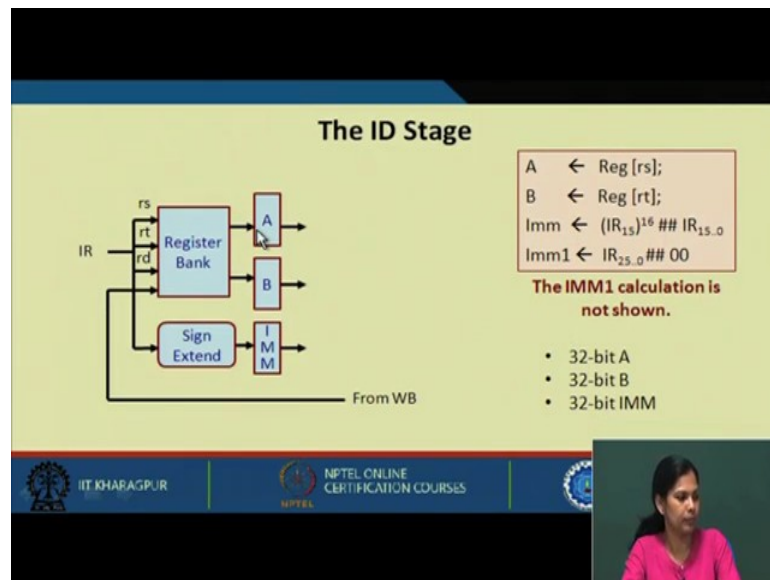
(Refer Slide Time: 02:04)



Now, let us see how the data path will be for the various stages. First take into consideration the IF stage. So, in the IF stage, from Mem[PC] we load the content of the memory location pointed by the PC. We will fetch the data and put it in IR; at the same time PC will be incremented by 4, and it will be stored in NPC. So, we require a 32-bit PC, a 32-bit NPC, a 32-bit instruction register, and of course, a 32-bit adder.

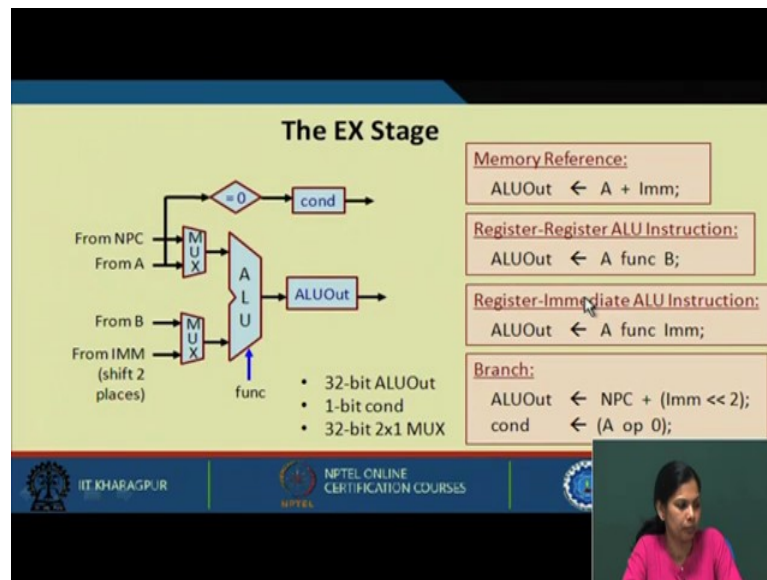
So, now we see that this PC will hit the instruction memory. PC contains the address of the next instruction to be executed. So, it will hit the instruction memory and the data will be available in IR, and this PC is added with a constant 4 and this is stored in NPC. So, these two operations can be performed using the following data path that is there.

(Refer Slide Time: 03:49)



Next move on with the ID stage. In the ID stage, what we said we decode that instruction, and after decoding that instruction we also fetch the particular fields that are the registers --- source register, the destination register, the immediate field, etc. and we make it available for that particular instruction execution. So, from the register bank this rs, rt and rd from IR it is coming, and from this register bank it is going to A and B; and after sign extension it is going to IMM. So, one of the source register values is coming depending on rs, another source register value is coming from rt, the immediate value which is sign extended to make it 32-bit if stored in IMM; Imm1 calculation is not shown; it is pretty similar.

(Refer Slide Time: 05:21)



Now, let us see what happens in the EX stage. So, we are looking here step-by-step fetch, decode, then execute, and for execute how the data path will be. For the execution we know that there can be various kinds of operations that can be present; one can be memory reference, another can be register-register ALU instruction, register-immediate ALU instruction, and branch instruction.

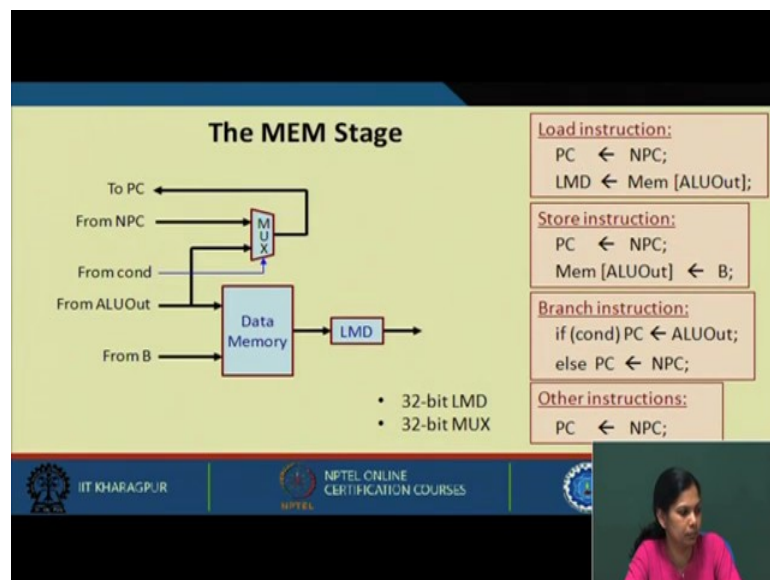
So, for all these kinds of instruction we see that this particular data path will be sufficient. We need to add with the immediate value. So, there is a MUX which is attached to this ALU is having two inputs, and in this ALU one is coming from NPC another is coming from A. So, depending on the select line this can be taken care which input will be going. So, for memory reference A will come in, accordingly the select line will get activated, and immediate value will be coming. So, here we can see that A and B are coming from these two MUX, and along with B that immediate value can also come in here, which is available after ALU operation. The ALU output is available at ALUOut. So, we are doing this operation and putting it in ALUOut. So, 32-bit ALUOut, 1-bit cond is required for condition checking, and 32-bit 2x1 MUX is also required.

Now, let us see for register-register ALU operation. So, if it is register-register then input A should go and B input should go; in that case the select line will be selected and A input will go to one input of ALU, and B input will go to another input of ALU, and the operation will get performed. Similarly for register-immediate ALU operation, A will

come in from the first MUX to one of the inputs of ALU, and Imm will come from another MUX and will go in into the next input of the ALU. And for branch the NPC value must be added with the immediate value by shifting it to left 2 positions.

So, in that case from this MUX NPC value will come in, and from this MUX this Imm value will go in, and the operation will be performed similarly for the cond. The cond will be available based on this A op 0. So, A will be operated with 0. So, input is coming, will be checked with 0, and the condition will be outputted here. So, all these operations are actually performed in the execute stage using the following data path.

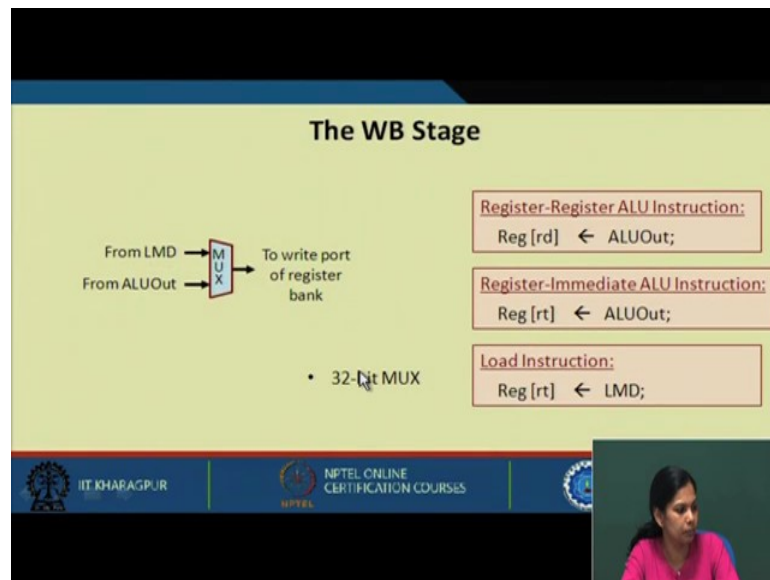
(Refer Slide Time: 09:30)



Let us move on Mem stage. In the Mem stage what basically happens which all instruction will require --- memory load operation, store operation, branch operation, and for all other operation NPC is stored in PC in this particular Mem stage. So, let us see the data path for Mem stage. So, what happens this is the data memory from where we will fetch the data, now we know that if we need to read it from a particular location we need to put that address in MDR basically and then we hit the data memory.

So, in this case ALUOut will be put in to the data memory, and then the value will be read and can be stored in LMD. So, for the load operation this happens where memory location pointed by ALUOut is read, and it is stored in LMD. And in all cases we can see that updated value of PC in NPC is stored in PC, so, similarly all other operation can be taken care using this particular data path.

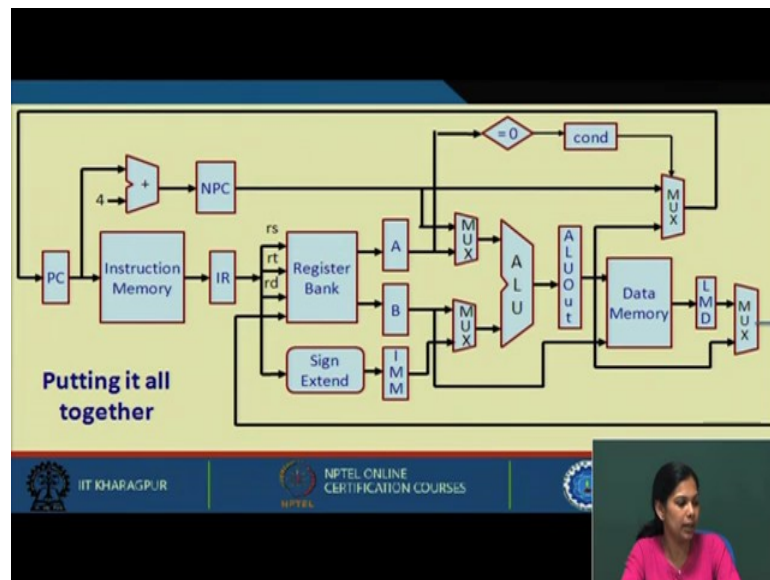
(Refer Slide Time: 12:04)



Let us move on with the WB stage. In the WB stage what happens, we write the value into the register. So, finally, whatever value; either that value is coming from the ALUOutput or the value is coming from memory, it should be written back into the destination register. So, in some cases rd is the destination register, and in some cases rt is the destination register. So, accordingly for register-register ALU operation, this ALUOutput is stored in rd; for register-immediate ALU operation ALUOut stored in rt. And for load instruction the LMD value will be stored in rt. So, this is how it can be done it can be selected from the MUX either from LMD or from ALUOut.

Till now we have seen that the various data path that is required in the various stages in IF, ID, EX, MEM and WB. So, in 5 stages we can see that; what is the data path that is required in MIPS.

(Refer Slide Time: 13:42)



Now, let us put it together all. So, whatever we have discussed starting from IF to ID then to EX and then to MEM and finally, to WB. This overall picture gives you an idea how it is actually happening. This is the data path that is there for MIPS. So, here initially the PC hits the instruction memory; the instruction is read and stored in IR. After it is stored in IR the instruction needs to be decoded; at the same time when it gets decoded some of the other register gets populated, that is, A, B, Imm, etc. And next we execute it. For execution it is required that we get the data from A and B, or from an immediate value; accordingly MUX are in place to select any one of the values out of the two which is fed to the ALU, and from ALU we are getting ALUOut.

Now, once the data is available in ALUOut it is some time required to get the data from memory. So, in that case it is again hit to the data memory, and the data is read and it is stored in a register known as LMD. Finally, from LMD it is written to the register bank in the WB step. So, all the steps that we have seen in bits and pieces in previous slides, here we have put them all together. So, this is the overall picture.

(Refer Slide Time: 16:20)

Simplicity of the Control Unit Design

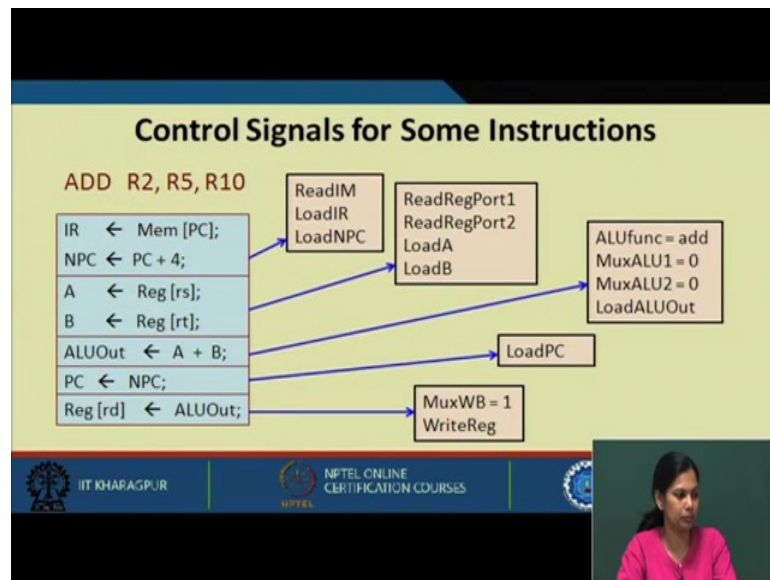
- Due to the regularity in instruction encoding and simplicity of the instruction set, the design of the control unit becomes very easy.
- Control signals in the data path:
 - a) LoadPC
 - b) LoadNPC
 - c) ReadIM
 - d) LoadIR
 - e) ReadRegPort1
 - f) ReadRegPort2
 - g) LoadA
 - h) LoadB
 - i) LoadIMM
 - j) MuxALU1
 - k) MuxALU2
 - l) ALUfunc
 - m) LoadALUOut
 - n) MuxPC
 - o) ReadDM
 - p) WriteDM
 - q) LoadLMD
 - r) MuxWB
 - s) WriteReg

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

Now, what is the simplicity of the control unit design here. You see that there is a regular structure here; that means, all the instruction we know are of fixed length; we know that this particular field is for this particular register, that is source or destination, this is an immediate field, and so on and so forth. So, we actually understand that for MIPS the instructions are very regular. So, all the instruction will be pretty much same depending on what kind of instruction it is. So, because of this due to the regularity in instruction encoding and simplicity of instruction set, the design of control unit becomes very easy.

So, control signals in the data path are as shown. Using these control signals in the data path, we can generate all the control signals for any instruction.

(Refer Slide Time: 18:40)

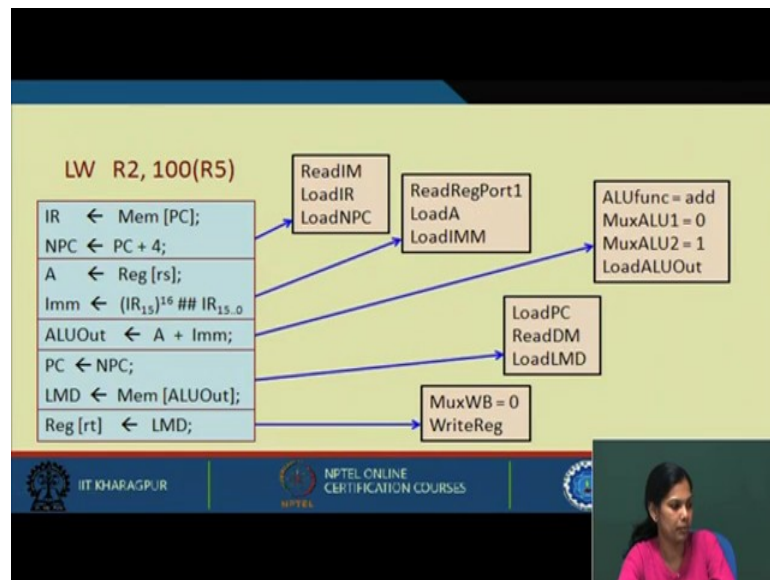


Now, let us see this control signals for some of the instructions. You recall our discussion for single bus architecture; we have seen how corresponding to the data path the control signals are generated, and how we can execute it. Similarly here also we have shown that these are the control signals available for MIPS. Now we will show how a particular instructions gets executed using those control signals. Take ADD R2,R5,R10. So, here we fetch the instruction: ReadIM, LoadIR, and then LoadNPC. So, in the first fetch step, these are the control signals that are activated.

Now, after doing, so, in the next step we are loading ReadRegPort1, ReadRegPort2 after reading from here load it to A and B, that is the decode phase. At this phase we are also decoding, and we are also fetching the value from the instruction and storing it in A and B. Now we are doing the exact ALU function. So, here ALUfunc will be add; this is add operation, MuxALU1 will be 0, we need to select A so MuxALU2 will also be 0, we need to select B and then after this operation LoadALUout. So, ALUOut will be loaded with the operation performed.

Now, here are the operations performed in the Mem phase. In the Mem phase we load PC. So, PC is loaded with the new value of the NPC and finally, in the WB phase the output of ALU that was present will be written to R2; in this MuxWB will be set to 1, and WriteReg will write it into the destination register. So, to execute this particular instruction these are the micro operation or control signals generated.

(Refer Slide Time: 22:09)



Let us see another instruction that is `LW R2,100(R5)`; that means 100 and R5 will be added then the value will be read from memory and it will be loaded in R2.

So, at different phases different things will happen. So, let us say this is the fetch phase. The fetch phase is standard. In the decode phase register value of `rs` will be loaded in `A`, and of course, other values will also get loaded, but here we are showing which are required for us which are needed for us. So, `Imm` will actually store the `IR` value 0 to 15 bit, and it is sign extended to make it 32-bit. We have to add this `Imm` with `A`, and we have to store it in `ALUOut`. So, `ALUOut` will have `A + Imm`; we add these two values and finally, this `NPC` will be loaded in `PC`. Now the content of `ALUOut` will be brought in from data memory, and it will be put in `LMD`; after this is done finally, from `LMD` it will be put into the destination register `rd` or `rt`.

So, let us see the corresponding control signals. So, first we `ReadIm`, `LoadIR` and `LoadNPC`. Similarly here we will `ReadRegPort1`, we will `LoadA` and you will `LoadImm`. Now you see `MuxALU2` will be 1; earlier when it was `A` and `B` the `MuxALU1` was 0 and `MUXALU2` was also 0, but now it will be the immediate value. So, in this case `MUXALU2` will be 1; the next value will get selected depending on the `MUX` select line and then we load. So, this will be added and will be loaded in `ALUOut`.

Next in the Mem phase of course, we will `LoadPC` that will be done for all the instructions, and now this `ALUOut` contains the address from where we have to read the

data. So, as this contains the address where to read the data we will read it from the data memory and we will load it to LMD. So, data is now read from the data memory and loaded in LMD in this particular phase.

And finally, we write back we write back the data into Reg[rt]. So, MuxWB will be 0 and we perform WriteReg. When we perform WriteReg, the LMD value is loaded in Reg[rt]. So, this is how we can perform the operation.

So, we have come to the end of lecture 22 and of course, week 4. So, in this week we have seen how we can design a control unit, the various methods that are present in the design of control unit. And by this time we have also shown that for MIPS what is the control unit, how we can design the control unit and so on.

Thank you.