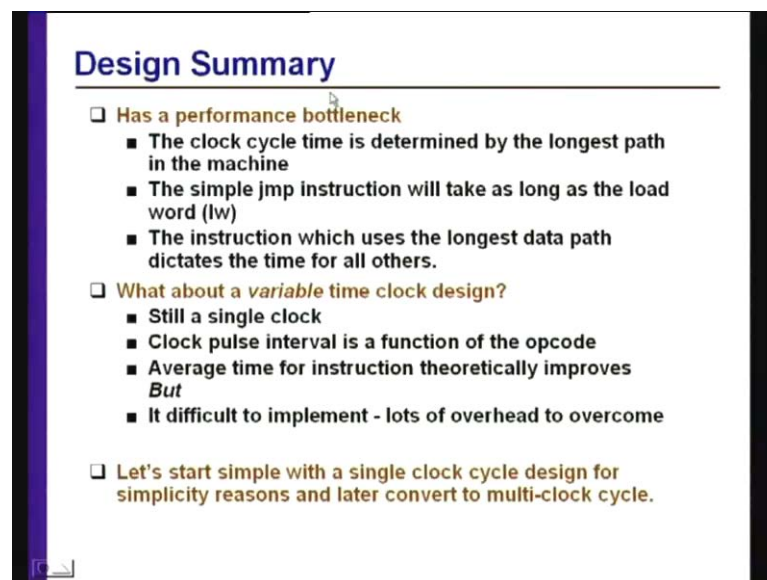


High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 5
MIPS ISA and Processor (Contd.)

Hello and welcome to today's lecture on MIPS Instruction Set Architecture and Processor, this is a second lecture on this topic. So, let us pick up the seg, where I left in my last lecture.

(Refer Slide Time: 01:14)



Design Summary

- ❑ **Has a performance bottleneck**
 - The clock cycle time is determined by the longest path in the machine
 - The simple jmp instruction will take as long as the load word (lw)
 - The instruction which uses the longest data path dictates the time for all others.
- ❑ **What about a variable time clock design?**
 - Still a single clock
 - Clock pulse interval is a function of the opcode
 - Average time for instruction theoretically improves
 - But*
 - It difficult to implement - lots of overhead to overcome
- ❑ **Let's start simple with a single clock cycle design for simplicity reasons and later convert to multi-clock cycle.**

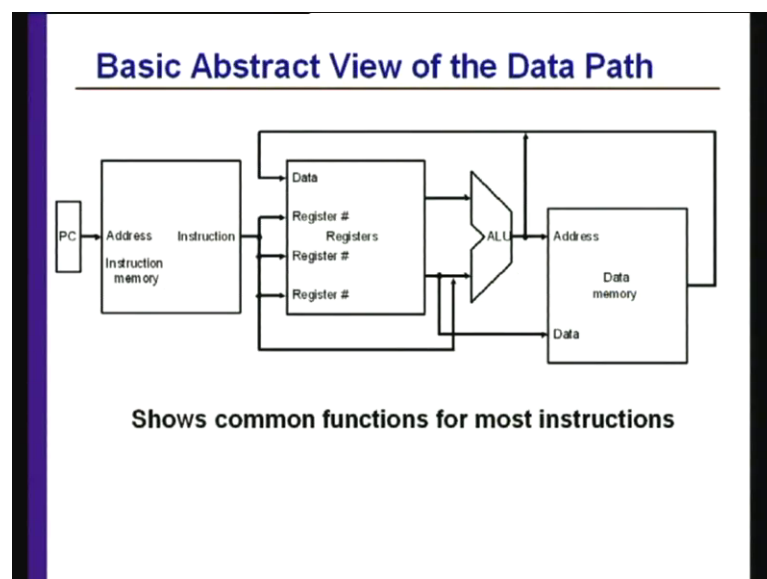
So, I concluded my last lecture with this design summary and where I explained, whenever you go for single cycle design, there is some performance bottleneck. You can go for a designing variable time clock design, but that will make the design very complicated, that is the reason why, we shall start with simple design with a single clock cycle for simplicity reasons. And later on I shall discuss about how you can make a multi cycle design, and also other complicated things like pipelining, you can incorporate instruction level parallelism and various other things.

(Refer Slide Time: 02:06)

- Data Path \leftarrow Functional Elements.
- Controller \leftarrow Control the functional Element

So, in your design, you will require two distinct type of components; one is known as data path, another is known as control path, controller you can say, sometimes it is called controller path. So, data path specifies the different functional elements that is required for performing computation and controller will control those functional elements. So, your processor will require data path and controller, so these are the two things we required.

(Refer Slide Time: 03:02)



So, let us see, what is the data path that is required for our MIPS processor, so this the basic abstract view of the data path. So, you start from the left side and so here this shows the most common functions, this is not complete, this is not a complete data path. As we progress in this lecture, we shall see, the other data path components that is required, so we were starting with the content of the program counter.

Program counter is holding the address of the next instruction that means, the processor will be having a special purpose register known as program counter and program counter will always hold the address of the next instruction. So, whenever you are turning the power on that time it is a responsibility of the operating system to load a proper value in the program counter. Subsequently, whenever you will be doing context switching, when you will be doing sub routine call, you will see that, the program counter has to be loaded with proper values.

However, whenever you are executing instruction in a sequential manner, the program counter has to be incremented by the 4, because your instruction length is 4 bytes. So, next instruction it will point, if you increment the program counter by 4. So, whenever you are executing instruction sequentially one after the other then everything that the controller needs to do is, to increment the program counter by 4, so that program counter will provide the address of the next instruction.

So, here is the instruction memory, from where the instruction will be fetched and then instruction will be available, instruction as you have seen, is a 32 bit instruction. And we have discussed about different instruction formats and if you perform arithmetical and logical operations, that source addresses will be applied directly from the instruction register. It will go to I mean, the operands will be taken from the instruction register and that register, this is the register bank that is, 32 registers is here, that 32 32 bit registers are here and you are applying the addresses of these registers.

So, operands will be available on this two arms of the arithmetical and logical unit, so arithmetical and logic unit will perform the required operation. Because, that instruction, it will also provide the operation to be perform, that later on when I shall discuss the control unit. That controller will be provide you, that instruction will be decoded and that controller will be provide you, the operation to be perform by the ALU. And ALU will

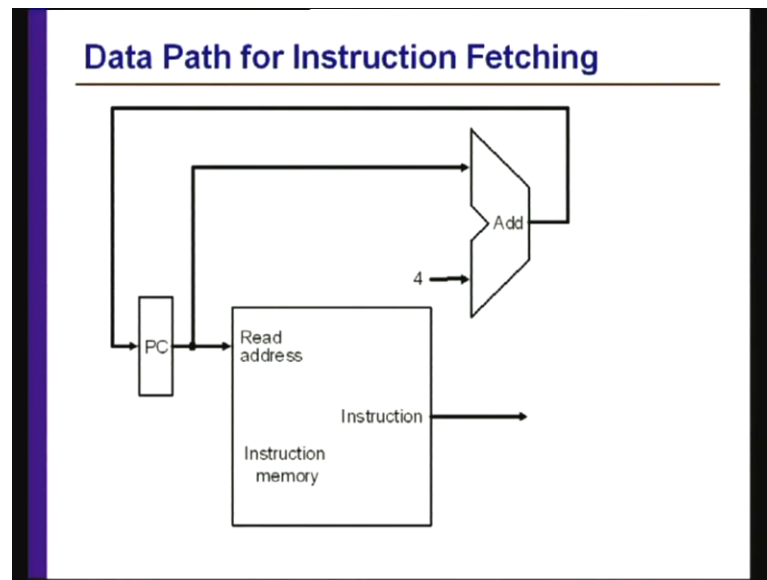
perform the operation and result has to be stored in the memory I mean or into the register.

If it is a R type instruction, that result will go back to the register, you can see data will be available here and the destination address is again provided by the instruction and result will be stored in the memory. So, if it involves that arithmetic and logical operation that means, data manipulation type of instructions then you require this register bank to supply the operands and also to store the result. So, you can see, it is 3 port register, 3 port register means, you can apply all the three inputs addresses.

And accordingly, two will be used to generate the output values from the two registers and one is for storing the result in the particular register. So, this is a 3 port register and two will provide the address, from where you will get the output and third one will be used to store the result, so that is the destination register address. And in case of load and store instructions, however you will be required this data memory. The data memory, whenever you are performing load, load means you are loading the value from memory into a register.

So, in that case, that address will be available here, ALU will compute the effective address. As you have seen, whenever it is a load type of instruction, that address will be generated by the ALU, effective address. And that address will be used to get the data from the memory and that data will go here and instruction will provide, in which the register data will be stored. Similarly, for store operation, again that address will be provided by this ALU and the data will be also available from one of the registers and that data will be stored in the memory location, for which the address is provided. So, in north cell, this is how different types of instructions are executed with the help of this data path. So, this is the data path you required for performing various arithmetic and logical operation, to perform load, to perform store and so on.

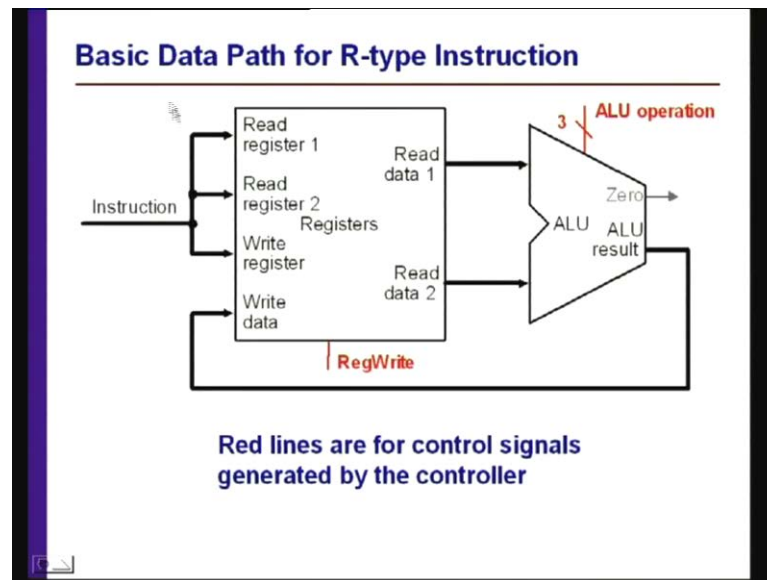
(Refer Slide Time: 08:59)



However, you have to extend it and let us see, how the different operations are performed. As I have already told, you will required another adder, why do you required another adder, because you know that, the next address has to be generated by adding 4 with the present value of the program counter. So, the program counter value is applied to this adder and 4 is applied to other arm and the PC plus 4 is applied to the program counter, so this is for instruction fetching.

So, each time you fetch one instruction, after the instruction is fetched, the new value of the program counter will be loaded I mean, new value will be loaded from this output of this adder. So, that is your PC plus 4 will be loaded in this program counter, which will provide the address of the next instruction, so this is for simple instruction fetching from sequential memory. Obviously, this does not perform that branch or jump, for that you have to add some more data path, that actually discuss later. So, this is for sequential fetching of instruction from the memory.

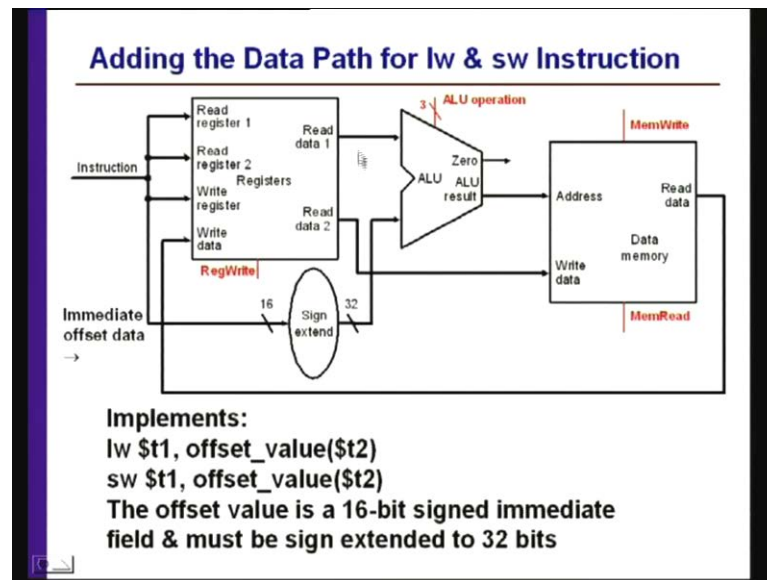
(Refer Slide Time: 10:13)



Then, basic data for R type instruction, as I was telling the operands will be available from the instruction I mean, operand addresses, registers addresses will be available. Then it will read data from the register and it will apply to the arithmetical logic unit, it will generate the result, that result will be applied into this write data input and that write register at this, again provided by the instruction. You have seen then in R type instruction, you have got 3 register addresses, two for address of operands and third is the destination address of the result.

And so all the three are applied here and data will be loaded in this and whenever you are doing this, you can see the signals will be generated by the controller. So, the controller will generate this register write signal and ALU operation signal to this ALU. So, here you required 3 bits assuming that, ALU can perform eight different operations addition, subtraction, multiplication and so on. And so these signals will be generated by the controller, so later on, we have to add controller to make the design of the processor compute, so this is on merely the data path.

(Refer Slide Time: 11:47)



Then, for loaded store instruction as I have already told, it will involve this data memory, so effective address will be generated in both cases by reading the data from one of the registers. And then that 16 bit data that will be coming as part of this instruction, that will be sign extended, I have already explained what did you mean by sign extension. The sign extended data will be added with the content of the register to generate the effective address.

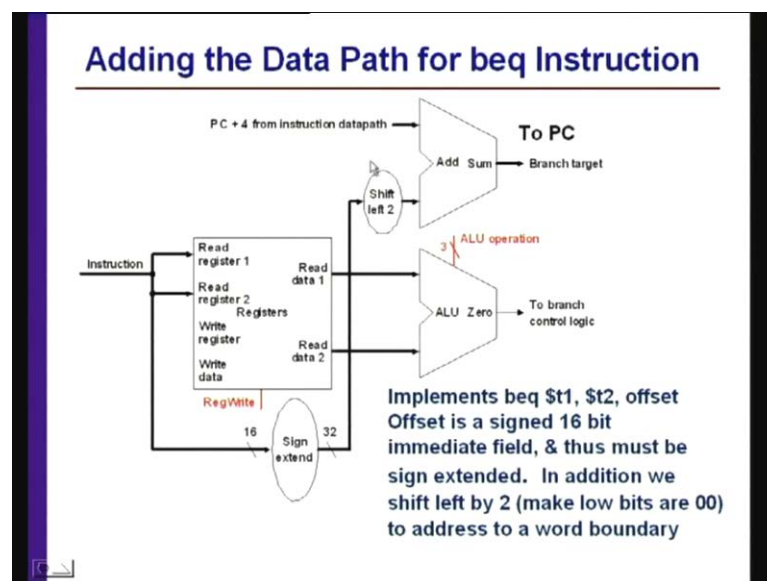
That address will applied to this data memory and this data memory will, you know in case of load, it will provide the data here. So, data will be available here and that will be written into the register, so this is that load typically, the value will be loaded into register in t 1. And this is the offset value is provided by t 2 that means, t 2 register is providing this offset and this is added with this and then you are loading this value in register t 1.

Similarly, this is the store word, store word means, that content of t 2, which will be available here, that t 2 will be available here, which will be written into the memory. So, that effective address is generated here again by adding the sign extended value with the content of t 2 register. So, do not get confuse with t 1 t 2, these are essentially the registers taken from the same 32 bit, that 32 32 bit registers. And in this particular case, you can see, the control signals to be generated by the controller for load and store instructions are given here.

Number 1 is register write, because that is the required when we are performing load instruction and that ALU operation will be addition and memory write, that will be required for load. Similarly, that will be required for store and memory read will be required load and for store, we will require memory write. Because, both are shown together that is why, you have got two signals, one of the two it will be generate at a time for load and store.

So, similarly here, register write will be taking place whenever you are performing load and for store, you have to read it and that data will be loaded into this register. So, here the offset value is 16 bit signed immediate field, must be sign extended to perform the addition. As I have already explained the need for sign extension and you can see here, how the sign extension is used to generate the 32 bit effective address, that is required to generate the address for the memory.

(Refer Slide Time: 14:57)



Now, we have come to another type of instruction that is, your branch if equal, so whenever you are doing branch if equal, two things are required. Number 1 is, whether the two register values are same or not, that decision should be known. If they are equal then branch will take place to a particular location and that address has to be generated. On the other hand, if they are not equal then of course the address is already known that is, PC plus 4, that is generated with the help of the register.

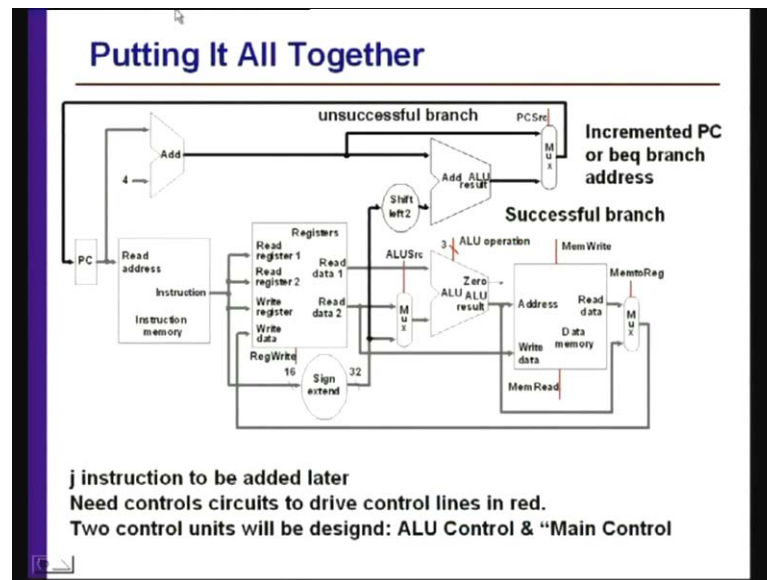
So, you can see here, you have used another adder here, which will generate that branch address. So, you are performing that sign extension, that you are adding with the content of PC plus 4 to generate the branch address. Because, that branch address has to be PC plus 4 next address, plus that you know with the plus some offset, that offset will come as part of the instruction and that will be sign extended and added with this value, PC plus 4.

So, this will be branch address, this will be generated I mean, this will be loaded whenever if the branch is taken that means, if it is equal. And who will decide, whether branch will be taken or not, so this is that ALU will perform subtraction of the two, content of two registers. So, content of the two registers will be applied to the ALU, it will do subtraction and if they are equal, result will be 0, if they are not equal, result will not be 0.

So, depending on that, that branch control logic, this will be applied to the controller, so controller will receive this signal and accordingly, it will generate control signals, whether depending on branch is taken or not taken. So, you can see, the control signals generated by the controller is register write, control signal generated is ALU operation in this particular case, subtraction. And this output of the ALU will go to the branch control logic that is, the controller.

So, this is the adding data path for branch if equal instruction, so branch if not equal that also I mean, same data path will solve the purpose. Implements branch if equal, t_1 and t_2 , $t_1 - t_2$ offset and offset is the signed 16 bit immediate field. So, this is the instruction `beq dollar t1 comma dollar t2 comma offset`, this is a instruction and offset is signed 16 bit immediate field and thus, must be sign extended. In addition, we shift left by 2, make low bits 00 to address the word boundary. We have seen that, word boundary has to be at the, should be multiple of 4, so that multiple of 4 means, the least significant bits will be 00 and that is what been done here.

(Refer Slide Time: 18:43)



Then, let us come to the complete data path, this is the complete data path, where which will perform all the different things. So, we see here, we not only required two separate memories, this is for reading instruction known as instruction memory, you will be required data memory, you will required the register file in any case. Then in addition to the arithmetic and logic unit, which will perform different computation, you required two adders, two separate adders.

One is performing that PC plus 4, whenever the branch is unsuccessful, this will be loaded into the program counter. On the other hand, when the branch is successful then that adder is generated by this adder. So, this adder will be generating that branch address and that will be multiplex and that program counter will be loaded by that branch address. So, you see, you require in addition to this adder, two more adders for the calculation of the two different branch addresses I mean, one is for unsuccessful branch, another for successful branch.

And if whether it is successful or unsuccessful, that is decided by the controller and controller will generate the signal to this multiplexer and accordingly, either PC plus 4 or the branch address will be loaded into the program counter. Then you can see here, different control signals, that is required in this particular case also. You will require another multiplexer to the second hand, because you will be either applying this value

here, that sign extended value to generate the address of this (Refer Time: 20:45) or you have to generate the operand value I mean, where the operand has to be stored.

So, either it will come from this sign extended form or it will come from the register, so you will required multiplexer that means, ALU source. ALU source is signal, which will be generate by the controller, either it will select this value or sign extended value, that will be applied to the ALU in different situations and this is the signal memory read or memory write, depending on load and store. And memory to register you can see here, again you will require multiplexer here, either the result is generated by the ALU, that has to be loaded into the register or that means, in case of load.

In case of load or store from two different sources it will come, either it will be come from the memory if it is a load or if it is store, it will come from here and it will be loaded into the register, that is why you will require multiplexer here. So, you can see, in addition to two additional adders, several multiplexers have been added in the data path. So, these are also data path component multiplexers, so you can see here, one multiplexer here, one multiplexer here, another multiplexer here. So, three multiplexers have been added as part of the data path to take care of the load and store and branches instructions.

(Refer Slide Time: 22:38)

Instruction	RegDst	RegWrite	ALUSrc	MemRead	MemWrite	MemToReg	PCSrc	ALU operation
R-format	1	1	0	0	0	0	0	0000 (and) 0001 (or) 0010 (add) 0110 (sub)
lw	0	1	1	1	0	1	0	0010 (add)
sw	X	0	1	0	1	X	0	0010 (add)
beq	x	0	0	0	0	X	1 or 0	0110 (sub)

And these are the various signals to be generated for our format and operation, the ALU operation is 0000 and 0001 for or and 00010 for add, 0110 for subtraction. So, here we

have restricted to and or add, subtract, so these are the four ALU operations, but since it has got 4 bits, there are apart from this four, you can have other ALU operations, which are not shown here for the sake of simplicity. Then for load word, the various control signal to be generated is shown here, register destination has to be 0, register write has to be 1.

Because, you are loading, ALU source has to be 1, accordingly the multiplexer will be selected, path will be selected then memory read has to be 1, memory write has to be 0 for load and then memory to register has to be 1, that multiplexer control signal has to be 1. Similarly, for store, they will be different, this is a material redundant then register write has to be 0, ALU source has to be 1, memory read has to be 0, memory write has to be 1.

So, it will be complement I mean, memory read, memory write as you can see, they will be complement to each other and then memory to register, it is irrespective of that redundant in this case. Then PC source has to be 0 then branch if equal you can see, all will be 0 and it is independent of the register direction destination and PC source, it will be either 0 or 1.

So, you can see and here in this case, it will be performing subtraction, because you know that, branch equal operation is performed by subtracting one operand from the other. And then checking whether they are equal or not I mean, result is 0 or not, so this is the R format ALU operation codes to be generated.

(Refer Slide Time: 24:54)

Control

- We next add the control unit that generates
 - write signal for each state element
 - control signals for each multiplexer
 - ALU control signal
- Input to control unit: instruction opcode and function code

Next, we add the control unit that generates write signal for each state element, control signals for each multiplexer, ALU control signal, input to control unit, instruction code and function code. So, so for what I have done, I have shown you the data path and also the various control signals that is required for different types of instructions. Now, we shall add a controller, so on top of the data path, you will require a controller and they took together will be implementing the processor.

(Refer Slide Time: 25:28)

Control Unit

- Divided into two parts
 - Main Control Unit
 - Input: 6-bit opcode
 - Output: all control signals for Muxes, RegWrite, MemRead, MemWrite and a 2-bit ALUOp signal
 - ALU Control Unit
 - Input: 2-bit ALUOp signal generated from Main Control Unit and 6-bit instruction function code
 - Output: 4-bit ALU control signal

And the control unit is divided into two parts, the main control unit, where the input is 6 bit opcode. That means, the main control unit will be having 6 bit opcode as the input and the output will be all control signals for multiplexers, register write, memory read, memory write and 2 bit ALU opcode signal. So, this will be the output and this will be the 6 bit opcode that is generated, that will be the main control unit function. With the input 6 bit of opcode, it will be generate this, depending on the opcode, it will generate various signals and I have already shown you, what will be generated.

Then, coming to the ALU control unit, input is 2 bit ALU opcode signal generated from the main control unit. So, I have seen that, 2 bit ALU opcode signal is coming from the main control unit, because here it will perform those four operations and or addition subtraction. So, these are generated by the ALU, by the main control unit and which will be applied to the ALU control unit as input and 6 bit instruction code also you applied to this, as input to the ALU control unit. So, 6 bit opcode along with two signals coming from the main control unit, will be applied to the control unit of the ALU. And accordingly, the control unit, 4 bit ALU output signal will be generated by the ALU control unit.

(Refer Slide Time: 27:11)

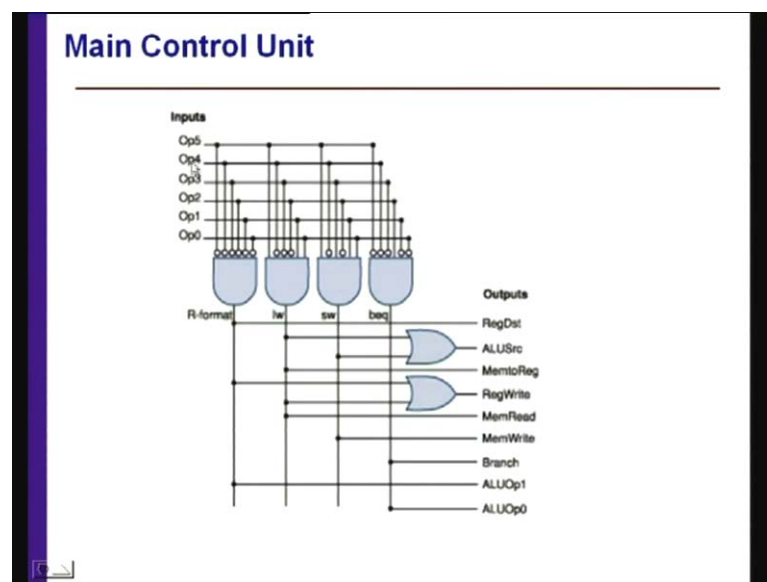
Truth Table for Main Control Unit					
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

So, let us see, what are the input and output, so these are the various inputs for the main control unit. I have already told that, there will be six, these are 6 bits coming from the opcode field, which be applied as input to the main control unit and it will generate these

signals, these are inputs, for our formats all are 0. For load word this is the signal, for store word this is the signal, for branch if equal this is the signal. And accordingly, the output that will be generated by the main control unit is shown here.

So, you can see, the register destination, ALU source, memory to register, control, register write control, memory to read control, these are the control signals generated. Branch, ALU of operation 1, ALU operation 2, these are control signals to be generated by the main control unit. And for different types of instructions, R format, load word, store word or beq, the different values to be generated by this main control unit is shown here.

(Refer Slide Time: 28:34)



And it can be realized by simple combinational circuit like this, so your input is 6 bits coming from the instruction code and it will generate various signals. So, I am not going to the design of this, you can find out from this table, this can be use as truth table and using this, you can realize a circuit. So, here you have many options, here I have shown, how beautifully different signals can be generated with the help of gates. Another implementation technique is by using (Refer Time: 29:13) PLA, Programmable Logic Array.

So, PLA can also be used as the controller, so PLA will receive input and it will generate control signals. So, what will be done and that I mean, that PLA can be used or this gates can be used, either way you will realize this main control unit.

(Refer Slide Time: 29:48)

ALU Control Unit

- ❑ Must describe hardware to compute 4-bit ALU control input given
 - 2-bit ALUOp signal from Main Control Unit
 - function code for arithmetic
- ❑ Describe it using a truth table (can turn into gates):

Now, let us focus on the ALU control unit, so this ALU control must describe hardware to compute 4 bit ALU control input given that means, the ALU, it will receive input from two sources, as I have already shown you. It will receive 2 bit ALU output from this here, these are two, ALU output 1, ALU output 0, these two will be applied to the ALU control unit along with the 6 bit opcode. So, the 6 bit opcode and the ALU control unit, those inputs will lead to the generation of this 4 bit ALU input control signals. So, it will compute the 4 bit ALU control input given, so 2 bit ALU from main control unit and function code from the arithmetic, so it will describe it using a truth table.

(Refer Slide Time: 30:46)

ALU Control bits

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

So, you can see, this is the truth table for that, so you can see, for different instruction codes, the ALU opcode generated by the main control unit is shown here. So, it can be 00 01 10, so three values, depending on different instruction load, store, branch equal and R type. So, we have restricted to register type of instruction, load and store and branch if equal. Obviously, this is not the complete instruction set, so you have restricted to the subset to generate the control unit.

Then, instruction operations are shown here, load word, store word, branch equal, additions, add, subtract, and operation, or operation, set on less than, R type. And this is the function field, function field that is coming from the, you know that, you may recall that, the instruction is having a function field. So, in addition to main opcode, there are three register fields and function field that means, you may recall that (Refer Time: 32:04), 6 bit opcode then there are three registers fields then function field and then here that shift field.

So, this function field is used here, in this particular case you can see, for generating the ALU control signals. So, ALU control signals, which has been generated, that 4 bit control signals generated are shown here. For add it will be 0010, for subtraction it will be 0110, for AND operation it will be 0000, for OR it will be 0001 and set on less than, that is been beq, it will be 0111. So, although 4 bits are there, but you can see essentially, one addition, subtraction, and or, and only five different values are required for the ALU control.

Because, either it will perform add operation or subtraction operation or AND operation or OR operation or set on less than. So, five different values are generated, although it has got four different fields. So, there are 16 possibilities, but five are used for the ALU control units.

(Refer Slide Time: 33:33)

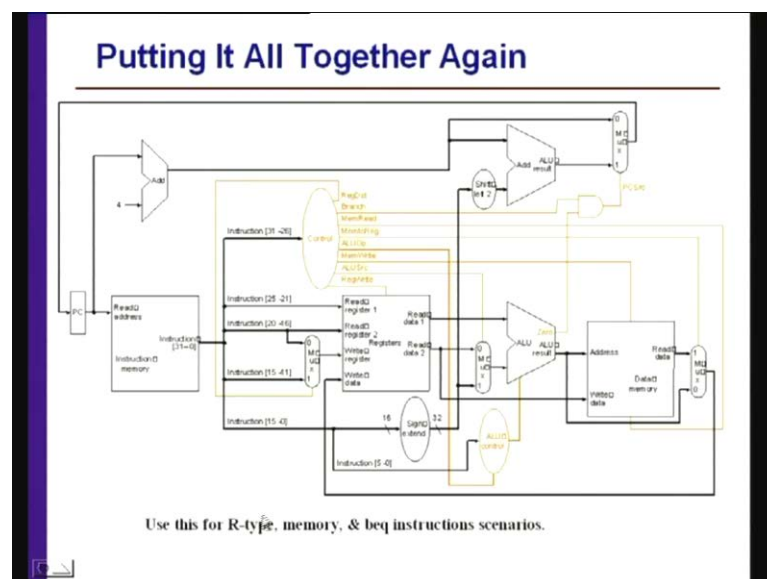
ALU Control Unit

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

FIGURE 5.13 The truth table for the three ALU control bits (called Operation). The inputs are the ALUOp and function code field. Only the entries for which the ALU control is asserted are shown. Some don't-care entries have been added. For example, the ALUOp does not use the encoding 11, so the truth table can contain entries 1X and X1, rather than 10 and 01. Also, when the function field is used, the first two bits (F5 and F4) of these instructions are always 10, so they are don't-care terms and are replaced with XX in the truth table.

Now, this is the ALU opcode field, this is the function field, same thing I believe the same thing is shown here. So, this is taken, this is the same thing represented different fields.

(Refer Slide Time: 33:51)



Now, I have put everything together that means, the function, the data path and the control path. So, you have got two controllers, main control unit and the ALU control unit, all are shown here. As you can see, these are the data path, I have already explain along with the different multiplexers then you have got, this is the main control unit,

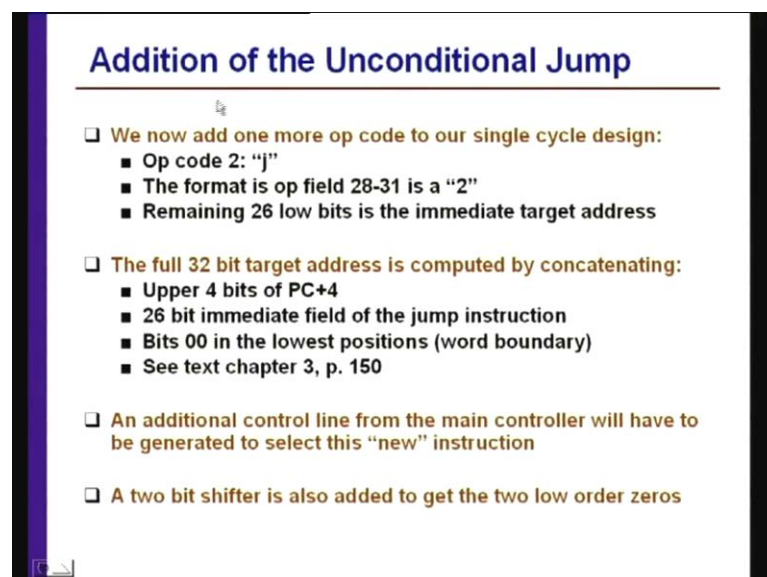
which will generate that register destination, branch. This will generate ultimately that PC source then it will generate that memory to memory read signal, which will come to this data memory then it has got ALU opcode to beq, it go to the ALU control unit.

Then, memory writes signal will go to the data memory then ALU source that will go to this multiplexer, whether it will come from register or come from this particular field, depending on that, multiplexer will select one of the two and applied to the ALU. So, this is the main control unit and this is the ALU control unit, ALU control unit is taking the 2 bit from the main control unit, 2 bits are shown here. And also the instruction that is, 0 to 5 main instruction code, that is also applied to the ALU control unit, that 0 to 5 field.

And that the 0 to 5, this is the field and this is applied to the ALU, that 4 bit control signal that is generated, is applied to the ALU. And ALU may generate, you know in case of beq, another additional signal that is required is 0 that is, the flag bit that is generated by the ALU or it will perform that add, subtract, AND, OR. The resultant outputs are, either it will be returning to the register, this will (Refer time: 36:03), it will go to the register; however for load and store, it will in for data memory.

So, you can see, this is the complete thing, which implements the R type instructions, where the two separate memories are shown, the beq instruction scenarios are depicted. And this is the complete data path and control signal, that is required to implement the subset of functions, that I have discussed.

(Refer Slide Time: 36:37)



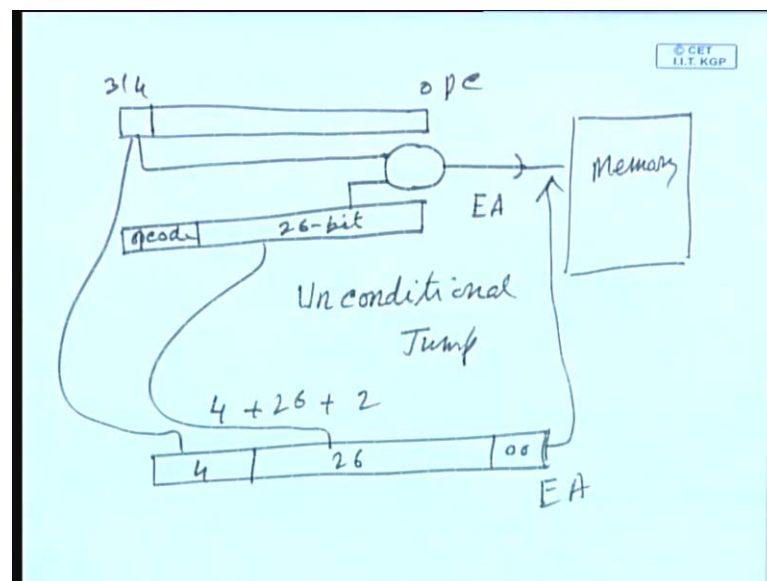
Addition of the Unconditional Jump

- ❑ We now add one more op code to our single cycle design:
 - Op code 2: "j"
 - The format is op field 28-31 is a "2"
 - Remaining 26 low bits is the immediate target address
- ❑ The full 32 bit target address is computed by concatenating:
 - Upper 4 bits of PC+4
 - 26 bit immediate field of the jump instruction
 - Bits 00 in the lowest positions (word boundary)
 - See text chapter 3, p. 150
- ❑ An additional control line from the main controller will have to be generated to select this "new" instruction
- ❑ A two bit shifter is also added to get the two low order zeros

Now, addition of the unconditional jump, earlier we have only added beq, now unconditional jump is also an important instruction that has to be added and how the data path and control unit will be affected is shown here. We now add one more opcode to our single cycle design, opcode 2 is j type and the format of the opcode field is 28 to 31 is 2 and remaining 26 low bits is the immediate target address. So, you can see that, pseudo direct addressing mode is used here.

So that means, whenever it is a unconditional branch then that pseudo direct addressing mode that means, that 26 bits field and the 6 bit opcode field, so that is been used here. And that 26 bits lower order bit field is providing the immediate target address, so this is used only in case of unconditional jump. And the full 32 bit target address is computed by concatenating the upper 4 bits of PC plus 4, so here the effective address generated in this way.

(Refer Slide Time: 38:07)



So, here is your program counter, program counter is 32 bit, 0 to 31 and your opcode in this case, the instruction format is 6 bit opcode and 26 bit is available that is, the immediate target value. So, what is done, this value and this 4 bit, so here you have got 26 bit and you require 4 bits from here. So, 26 bit is immediate field and 4 bit for the PC plus 4, they are concatenating, so these two are concatenating, not added. So, these two are concatenating to generate the effective address, in case of your unconditional jump.

So, an additional control line from the main controller will have to be generated to select the new instruction and 2 bit shifter is also added to get the two lower order 0s. So, this can be very easily obtained by shifting this 26 bit and to show that, two 00s are inserted.

Final Design with jump Instruction

The diagram illustrates a CPU architecture with the following components and connections:

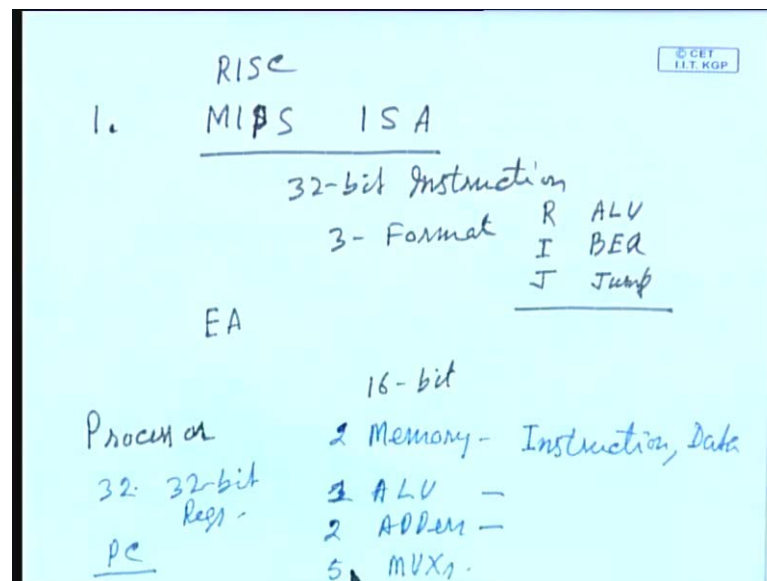
- PC (Program Counter):** Receives the current instruction address and calculates $PC + 4[19:16]$. It branches to a new address if the jump instruction is taken.
- Instruction Register:** Receives instructions from memory and provides them to the ALU and control logic.
- ALU (Arithmetic Logic Unit):** Performs operations on register data and provides a zero flag. It receives control signals like 'jump' and 'branch'.
- Data Memory:** Handles read and write operations. It receives address and data from the ALU and provides data back to the ALU.
- Control Logic:** Manages the execution of instructions, including the jump instruction. It receives control signals like 'jump', 'branch', and 'ALU result'.

The jump instruction is implemented by checking the 'jump' bit in the instruction. If set, the PC is updated with the target address from the instruction's immediate field. The ALU result is used to set the zero flag, which is used by the control logic to determine if a jump should be taken.

So, you can see here, those things not changed, you will require the instruction memory, you will require the data memory, you will require the main ALU, you will require two more adders for generating the addresses. Then you will require number of multiplexers.

one here, one here, one here and another two, only new is the additional multiplexer, which is required for this last jump type instruction. So, this is the final design, showing the data path and the two controllers that is required for realizing the processor. So, in this lecture, let me summarize what I have discussed in this lecture, I have first introduced to you in two lectures rather.

(Refer Slide Time: 42:20)



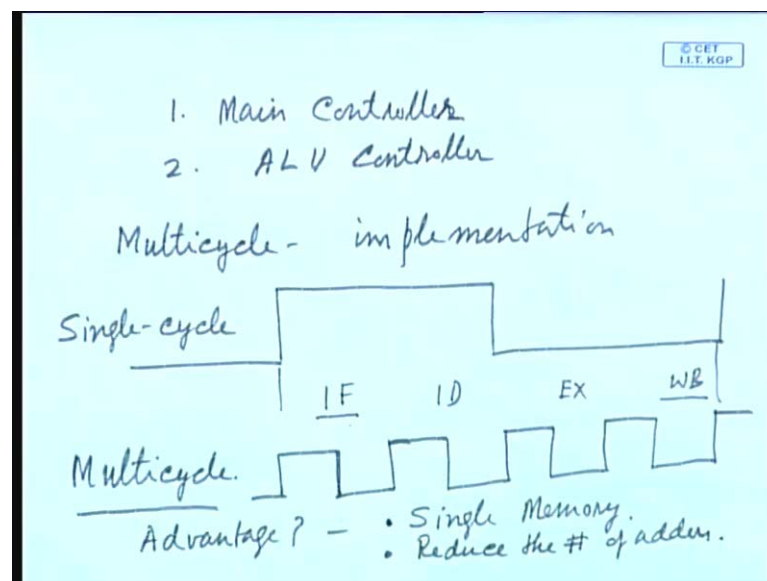
In two lectures I have included, I have discussed the MIPS instruction set architecture and of course, I have only discussed a subset of the instruction set architecture of MIPS. So, MIPS is a processor, risk processor, which has got single instruction size, 32 bit instruction with three different formats that is, your R type, I type and J type. And R type is used for conventional, you know ALU operations addition, subtraction, AND, OR like that.

And I type instructions are used for branch if equal and J type instruction is used for jump and accordingly we have seen, how the effective address is calculated in different situations. In this particular case, beq that 16 bit offset, 16 bit that is provided as the part of the instruction is sign extended and added with the register to generate the effective address. And in case of this jump type instruction, we have seen, how this is done by concatenating different fields, 4 bit is coming for the PC, 26 bit coming from the instruction and by shifting it by 2 bit, we are getting two 0 bits.

So, that is how the effective address is generated, whenever the jump type instruction is added and to incorporate these MIPS size to implement this MIPS size, we have seen that, you will require your processor. For processor requires, what are the things requires, it requires two different types of memories, one is for instruction and second one is for data. You will require three I mean, rather one ALU and two adders, two adders and one ALU is required.

And by doing that, you can implement all the data paths and of course, you will require a number of multiplexers. You will require how many multiplexers, you can see here 1 2 3 4 5, 5 multiplexers and of course, in addition to 32 bit, 30 to 32 bit registers, you require another special purpose register that is, program counter and this is the data path that is required.

(Refer Slide Time: 45:44)



And we have implemented the controller by using two different controllers, number 1 is main controller and second is the required, we have used one ALU controller, so main controller and ALU controller together controls the entire data path. Now, before I conclude, let me discuss about the multi cycle implementation, so this was the single cycle that was used to perform four different things. What are the four different things, instruction fetch, instruction decode, ALU operation and write back.

So, all the four instruction fetch, instruction decode, execution of instruction and write back, all the four were performed in a single cycle, in one go using a single clock.

Instead of that, what you can do, you can have four different clocks, say you make it multi cycle. So, in one cycle, you perform instruction fetch, in another cycle you use instruction decode, in another cycle you perform execution and in another cycle you use write back.

So, you can see, this is your single cycle and this is your multi cycle, so whenever you go from single cycle to multi cycle obviously, the clock frequency will be higher. So, in this particular case, it will be four times that of the single cycle, because we have of course, assume that, instruction fetch, instruction decode, instruction execution and write back, all of them requires the same time, but in reality, they will not be same.

So, again the frequency will be decided by the slowest of the four different operations, when this case for simplicity, we have assumed all of them requires the same time. Now, whenever you go for multi cycle, what is the advantage, say we have seen a single cycle, we will require so much of hardware, we have already seen that, two different types of memories, one ALU and two adders and so many multiplexers. Now, whenever you go for multi cycle, do you require so much of hardware, can it be reduce, the answer is yes.

How it can be done, number 1 is, you do not require two different separate memories, because we have seen that, in two different cycles whenever you will be performing instruction fetch, you will not be doing write back. So, you can have a single memory, in fact in the earlier years, when the computer was I mean, processors are designed, they were having a single memory and they are known as Princeton architecture. Only subsequently, to facilitate pipelining and other thing, you required two separate memories.

So, if you go for multi cycle, of course not using a pipelining then you can have a single memory, so which is the done by Vonum architecture or Princeton architecture. Second thing is, you can reduced some of these other function unit, because you see when you will be doing instruction fetch, you will be using this adder, when you will be performing execution, you will be using this ALU. So, possibly the same ALU can do this addition, if you go for multi cycle.

Similarly, when it will be calculating the address depending on the branch, this adder also can be perform by this that means, some of the functional units can be reduced, whenever you go for the multi cycle implementation. So, however that means, reduce the

number of adders, the question is, what is the disadvantage? In this world, nothing is one sided, it has defiantly got some advantage, what is the disadvantage, is there any disadvantage.

Whenever you go from single cycle to multi cycle, the disadvantage is that, controller is very complex, not only controller is complex, you have to address some additional functional units like multiplexer and other thing. So, your design of the data path and the controller will be very complex, whenever you go for multi cycle implementation. So, with this, let me conclude this lecture, I have discussed in detail the single cycle implementation of the MIPS processor and given some overview about multi cycle implementation. In my next lecture, I shall discuss how you can implement pipelining.

Thank you.