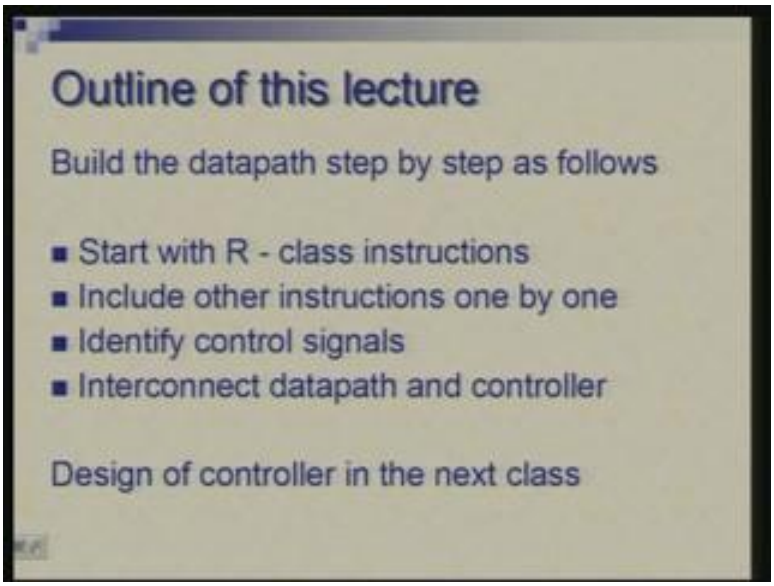
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 18
Processor Design (Contd..)

In the previous lecture we had looked at the basic building blocks which are required to construct the processor design. We will now try to put these together to come up with a very simple design and later on we will look at the performance issues and try to improve the design. So what we will do today is have a simplest possible solution to the problem of taking a set of instructions and having a circuit to execute those instructions. So we will build this design in small steps that you can see each and every step clearly and get a clear picture of how the circuit is getting designed.

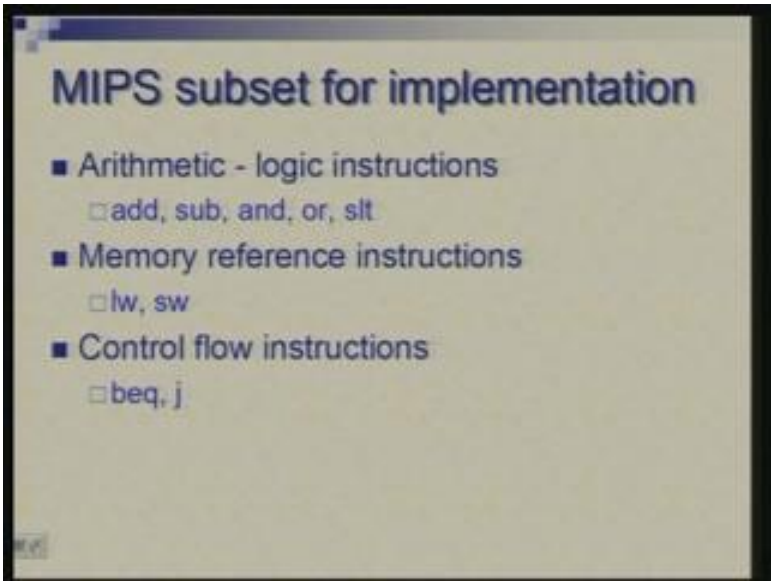
We will start with small set of instructions. First we will take only R - class instructions which includes arithmetic, logical and comparison; add, subtract, AND, OR and slt. So, that will be only part of the solution then we will add instruction by instruction and see how the whole thing can be built. So, with the basic skeleton of the design we will include other instructions step by step. So first we will include load store instructions to the basic set of four five instructions and then we will include the jump and branch instructions. So, after having put the data path together we will try to see how you control it; what actually is required in terms of control signals to make it do the right operation at the right time
and we will interconnect a controller to this data path which we will build. We will not go into detailed design of the controller; that we will take up in the next class.

(Refer Slide Time: 02:50 min)



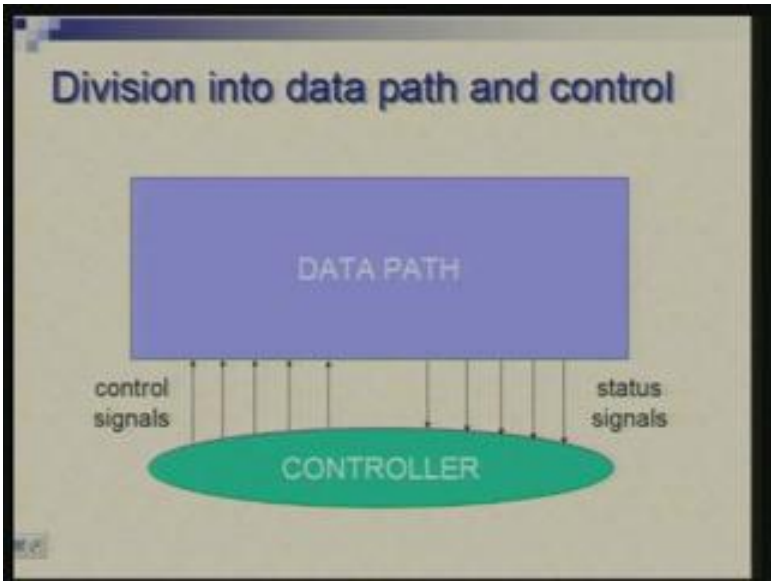
So, once again let us look at the subset of the instructions which we have set out to consider for building this data path. Among the arithmetic and logical instructions we have this five add, subtract, two arithmetic instructions; AND, OR two logical instructions and slt which does the comparison. So they are all of same class in the sense that they take two operands from registers, perform some operation, put the result in register. Then we come to load and store which access memory. So basically the data is transferred between register file and memory one way or the other and then instructions which influence the flow of control beq which does a comparison and then decide which where to go and jump which is an unconditional jump instruction.

(Refer Slide Time: 03:32 min)



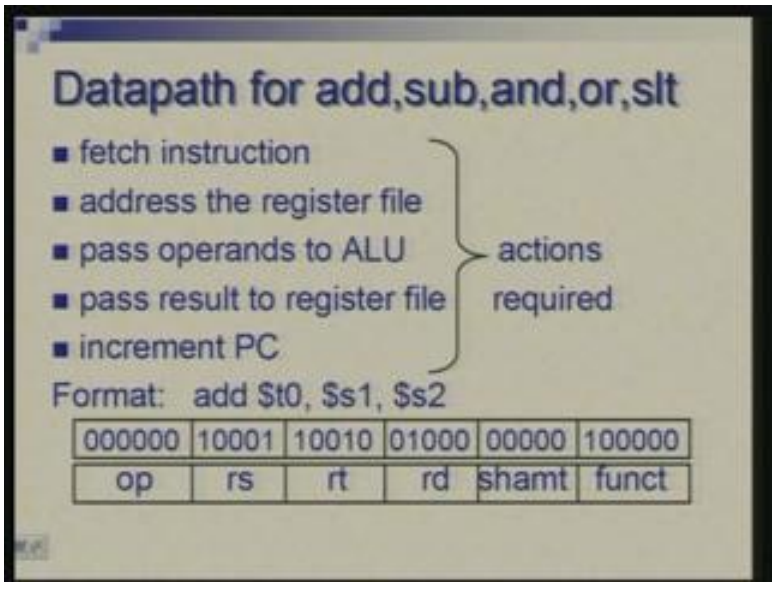
As I mentioned in the last class the whole design will have two parts: the data path and the controller. The signal which go from controller to the data path are called control signals and the signal which comes from data path to the controller are considered as a status signal. So controller times the activities in the data path and also directs what has to be done in which clock cycle or which instant. So, the status signal is information which controller seeks from the data path to decide its actions.

(Refer Slide Time: 04:22 min)



Now we are going to begin with these five instructions: add, subtract, AND, OR and slt. The process involve or the actions which will be required would be to get the instruction from the memory, taking program counter contents as address then depending upon the fields which contain register values we access the register file. So the register addresses will come from the instruction fields and the operand which we get from register file are passed on to ALU, then the result produced by ALU is passed on to the register file and we also increment the PC and make it ready for the next instruction.

(Refer Slide Time: 5:10)

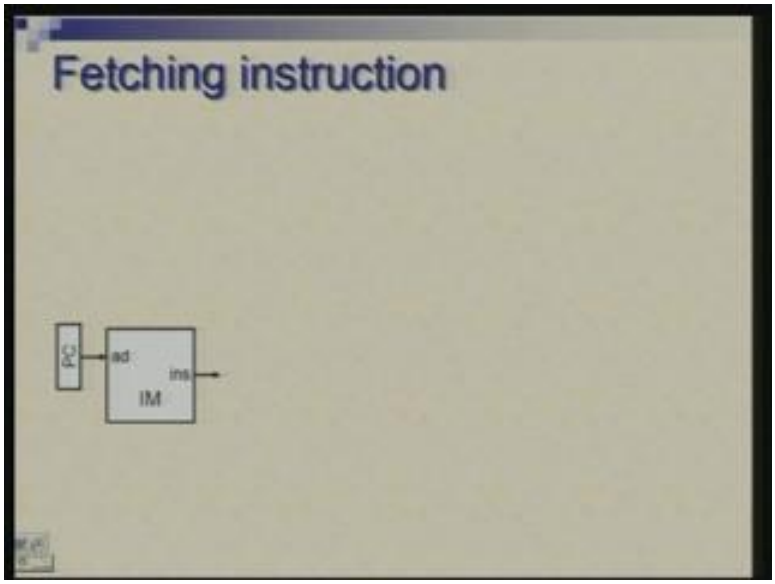


So, before we begin let us look at the format of these instructions. The format has several fields; a 6-bit field is the opcode field and for all these instructions this is common. There is a single code which actually defines this; a larger group than this (Refer Slide Time: 5:34) this is only a subset of all the instructions which have a common opcode here, then there is a source register 5-bit, third register 5-bit these two are the operands these two specify addresses of the operands and another 5-bit field specifies the address of the destination. So all these are numbers from 0 to 31 and specify one of the registers. Then this is unused (Refer Slide Time: 6:01) in some instructions this is used to specify shift amount and it is this field which is called function field which will distinguish these five instructions from one another and also from other instructions which are part of the group. So we would need to look at all the fields except for this field which is just to be ignored (Refer Slide Time: 6:25)

So the action begins by fetching an instruction from the program memory. We have a program counter, a register which will carry the address of the current instruction and there is an instruction memory. Instruction memory in this design will be assumed to be having fixed contents, we are not going to change the contents so the only input to this is an address input and the only output is an instruction which come out of this. So you given address instantly the instruction comes out; we assume that the program is somehow stored in this memory already by some means.

In fact such memories are called read only memories if you are familiar with where by a special process you load the contents in the memory and then you can only access it you can read it. So, in that sense this will behave like a combinational circuit; does not require a clock, so you given an input you have an output but the function which transforms input and output is fixed and by a special process it can be changed. Therefore, the PC feeds the address input of this memory and we get an instruction.

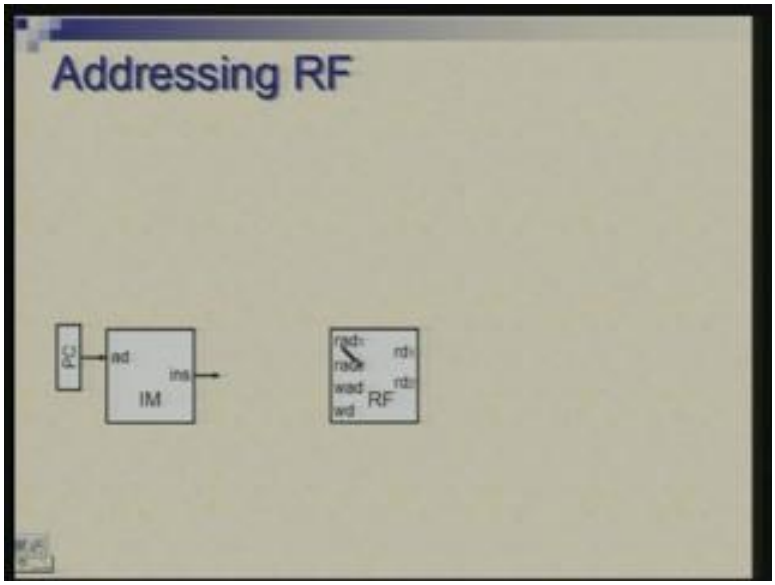
(Refer Slide Time: 7:51)



The next thing is to look at the instruction; various fields in instruction and address the register file. So, particularly for these five instructions we are talking of, we need to access rs and rt which was in second and third field in the instruction if you remember and bits of this 32-bit instruction which we are getting would be used to address register file. So register file as I discussed yesterday is specifically for this particular design; we need register file as an array of register with a provision of reading two values at a time; you can read two registers and you can write one register at any given time.

So now the addresses for these three things: two readings and one writing is provided independently. They could be in general different, two or more also can coincide but a register file will respond correctly in all these cases.

(Refer Slide Time: 9:03)

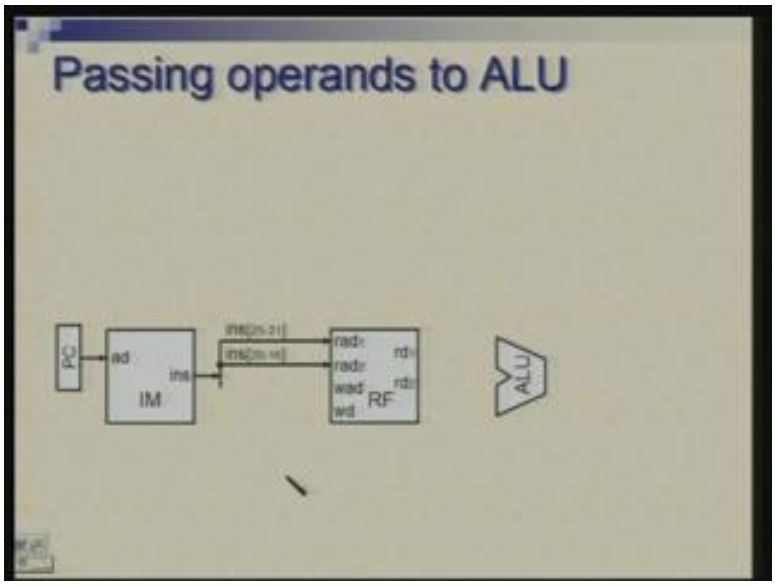


The three address inputs are read address 1, read address 2 and write address; these are the three inputs, each of these are 5 bits, there are two data outputs: read data 1 and read data 2 and there is one data input write data. Hence, we have instructions which is 32 bits so specific fields are being tapped out of this. From bit number 21 to 25 this is rs the source register and that goes to one address; bit number 16 to 20 forms a field which defines rt or the third register, this goes and addresses rad2 or the second read address. So, out of these 32 bits we are taking two groups of 5 bits and connecting it to the register file.

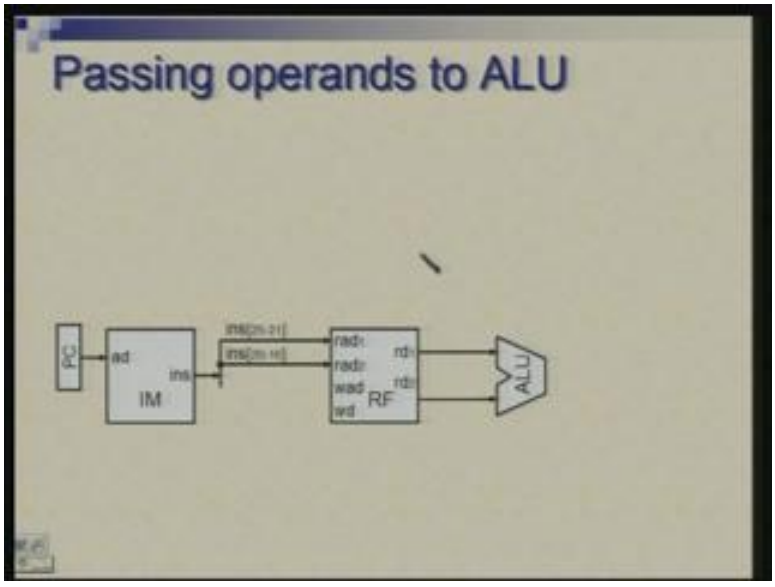
Now, once addresses are given this register file will respond with data here (Refer Slide Time: 10:12) and the operand which will come out of this will need to be passed on to the ALU. So we have an ALU; imagine the same design which we did couple of lectures back where, by specifying some control signal you could perform addition, subtraction, AND operation, OR operation you could do comparison for equality, you could do comparison for less than for the purpose of slt so same ALU we are putting now as a

block box, we are not looking into details of what is inside, we understand that design and we are simply using it to build a larger circuit now.

(Refer Slide Time: 10:20)



(Refer Slide Time: 10:50)



So these two outputs are forming two inputs or two operands for the ALU and next the result which is produced by ALU would be sent to register file for storage and that is where the cycle of flow of data or cycle of instruction would be complete. So now, at the moment I am not worrying about how to control ALU to do the right function. Now we are looking at those five instructions together and ALU would need to be told which of these instructions is. So we will, eventually when we talk of control we will look at bit

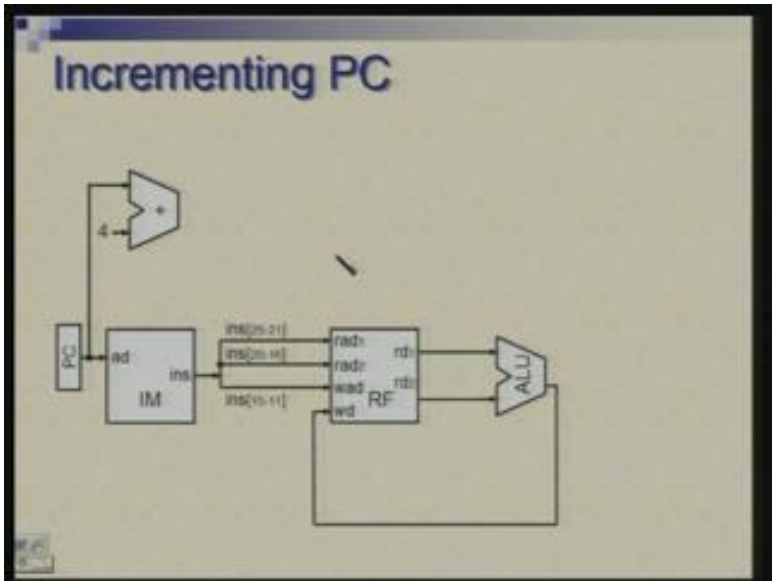
number 26 to 31 the opcode field and bit number 0 to 5 bit number 0 to 5, the function field we will look at those fields and then pass on some signals to ALU so that it does the right operation.

Right now we are not distinguishing between those fives but just looking at the overall flow of the data. The output of the ALU goes back to this register file to the right port (Refer Slide Time: 12:03). Let me just emphasis the terminology; these are called ports for the register file; port meaning there is something like a gateway so there are two read ports in this and then one write port and this is going back to the write port.

Therefore, now when we are writing we also need to arrange for the address where it has to be written so we need to look at another field from here and make sure that the address is also delivered correctly to the register file. So, bit number 11 to 15 are the destination address and that connects to the third address input. So now with this the cycle is complete. So basically starting with PC we have a sequential element here so at the edge of the clock a new value is available at the output of PC which defines address of a new instruction and as a function of that we get the instruction, as a function of that we get these operand, as a function of that we get the result and finally the result is available at the input of the register file.

Therefore, now at this point when a clock edge comes to the register file this information will get stored at that instant. So we are assuming that transition in the state of register file would be edge triggered. So you have one clock at which PC gets the value and at the next edge of the clock the result of this instruction will get stored in the register file and at the same time we will see in the next slide that PC will have to get a new value and be ready for the next instruction. So let us a complete that part and see how PC is to be incremented. So all we need to do is have a 4 added to the PC contents and connect the result back to the input of PC.

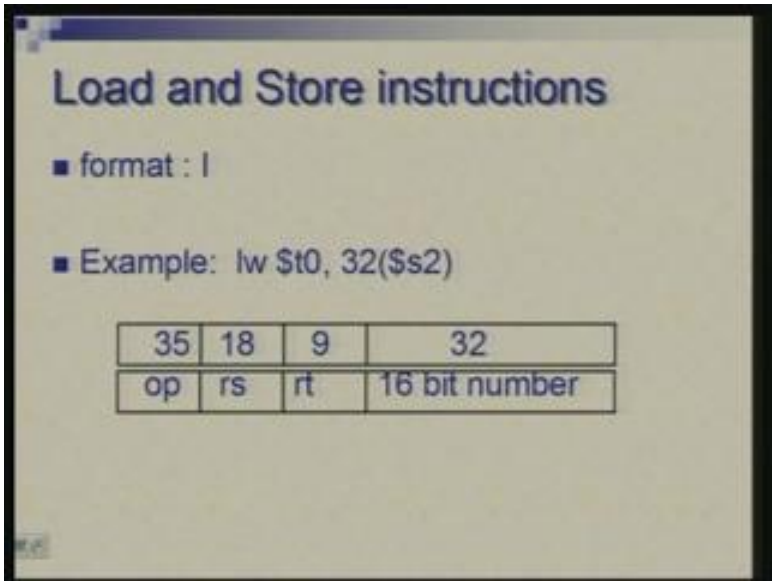
(Refer Slide Time: 14:10)



This is an adder with one input as 4 (Refer Slide Time: 14:13), one input as PC and output of this is going back to PC. So you give, let us say, at time t the output of PC was available for a new instruction and then at time t plus 1 I am counting time in terms of clock cycles not in nanoseconds and all. So at time t plus 1 which is the next clock you get a trigger here and trigger there so the current instruction completes by storing its value storing the result in register file and at the same time PC gets the new value and is ready for the next instruction. So that completes one cycle and if the next instruction was also of the same type then in the next cycle that instruction will be executed. So this cycle can go on and the important thing to note here is that the instruction is executed in one clock cycle.

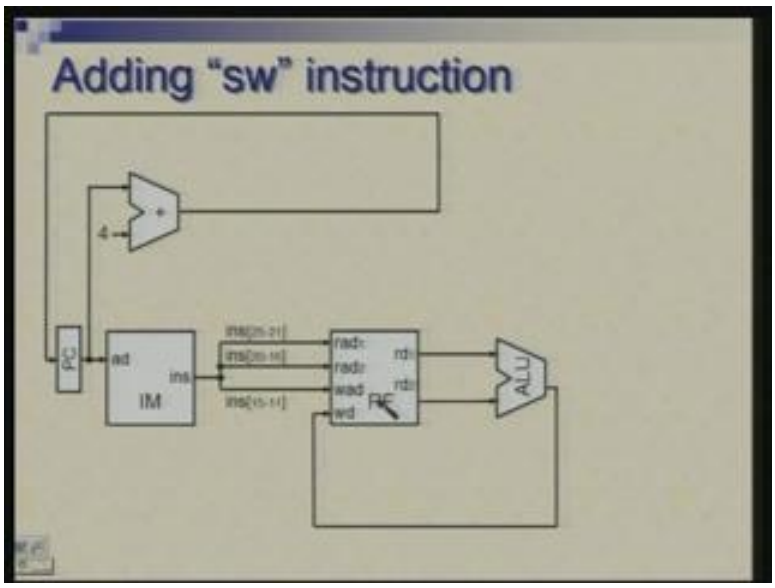
Remember that in the past we have talked of CPI and we have talked of figures of CPI which are 2 3 4 and so on; so later on we will discuss design where instruction execution does take multiple cycles and there are good reasons why we would do that but currently this design is aimed at doing the instruction in a single cycle and that was the simple component which I was talking of; we are talking of simple possible design.

(Refer Slide Time: 15:47)



Now we have seen the design for these five instructions. We need to now go further; we need to augment this to include more instructions and we will see that by making small incremental changes we can accommodate more and more instructions. So let us address load and store instruction next. The common thing there is that memory has to be accessed. And the mechanism for calculation of memory address is same. You have to take this number (Refer Slide Time: 16:25) which is the signed offset 16-bit number and contents of this register, add the two and apply that as address for the memory and this field defines the register which will exchange the data with the memory depending upon whether it is load or store. So we need to look at these two registers specified by these two fields and a constant.

(Refer Slide Time: 16:55 min)



Here is a same design we have done so far and now we will add more things to it to make it possible to do store instruction we will then add load instruction. So first of all we need to bring in data memory. Unlike instruction memory this will have data input as well and that is actually the only difference. You have address, read port, write port. Well, actually strictly speaking it will not be considered as two port memory it will be considered a one port memory because you either do read or write there is a single address and we will define control signals which will ensure that either you perform a read or write; so it is a single port memory but it can do read or write whereas this is a three port memory (Refer Slide Time: 17:48) two read ports and one write port.
This one has a single read write port.

In fact some memory modules have these in common. There is a same set of wires which are connected to memory through which you can send the data in or take the data out so that data terminal is a bidirectional one; there is one address line and a bidirectional data line. In this case, of course, again for simplicity we have separate read and write lines but there is a common address line. This is also a single port memory (Refer Slide Time: 18:23) with a single read port and that is all.

Now we have positioned the memory here, we now need to connect the inputs and outputs for this. So, first of all the address as I mentioned will be produced by performing an addition. So we will use ALU for doing the same thing. Because load store instruction does not require other arithmetic operations to be performed we will use this ALU itself to calculate the address. For doing PC plus 4 we could not used this because we had instructions which were using for some other purpose and this PC plus 4 was being done over and above all that so we required a different piece of hardware here. But for load store instruction we will use ALU to do the address calculation and therefore I have connected it in this manner.

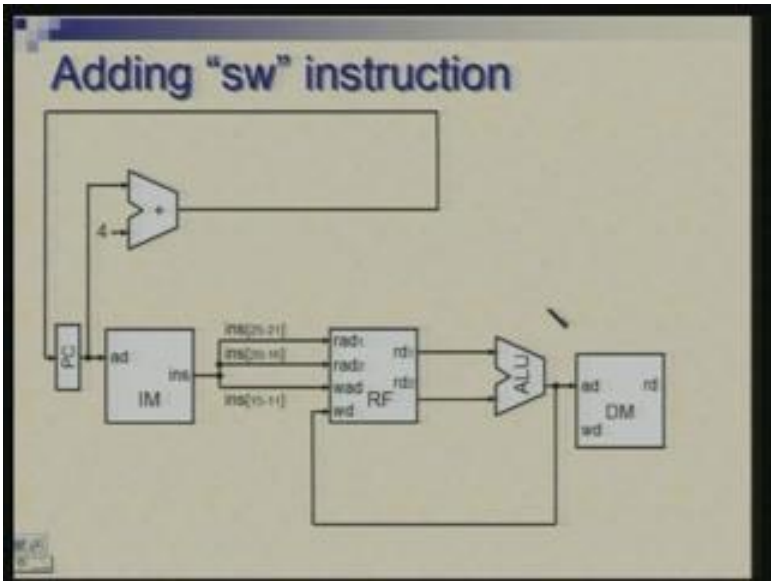
I need to make sure that the right inputs come to the ALU for calculating address and that is the next thing. So we do not directly connect register file output to ALU because for address calculation the constant coming from instruction will be loaded; we need to put a multiplexor here which will have this rd2 output for R - class of instruction but it will have something coming from this for performing address calculation. I have not connected it yet here because those 16-bits which we get from instructions need to be sign extended before we can pass it on to ALU. this is the block which is doing signed extension (Refer Slide Time: 20:16), it takes 16-bits as input, bit number 0 to 15 of the instruction and it does sign extension so here it is a 32-bit output.

Wherever 'I have not labeled things' assume that you have 32-bit output 32-bit signals. Each wire is actually carrying 32 bits. Exceptions are here where I have labeled explicitly or here I have also indicated that there are 16-bits explicitly. So now you see what is happening that rs specified by this field would bring out some 32-bit number here; 16-bit taken from here sign extended and we will control the multiplexor to select this path.

When it is load or store it is this path which will be selected (Refer Slide Time: 21:17) so we must give input one to the multiplexor, control input one and for add subtract instructions we must give zero so that this goes in. So, wherever we have actually two paths converging to the same destination you will notice that we have put a multiplexor and then it is the responsibility of controller to control this multiplexor correctly so that depending upon what we are trying do the right thing gets done or the right data gets passed through that multiplexor.

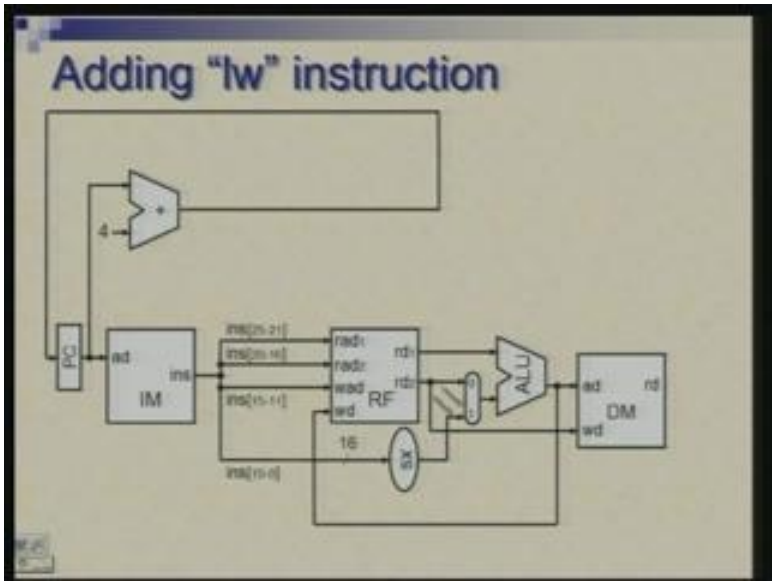
This part (Refer Slide Time: 21:52) takes care of inputs for data memory, address input in particular so it comes from here and inputs of the adder inputs of the ALU are also arranged now properly. To complete the picture we also have to make sure that the data from register file goes to the data input of this memory. So where does it come from? In this diagram where will the data come from which will have to be connected to this wd? [Conversation between student and Professor (22:25)…….Yeah, rd2…..] because it is a rt the third field which actually specifies which register has to be written so, that is already happening here and that output is available here; we simply need to connect this to this.

(Refer Slide Time: 22:44 min)



Now this is the complete arrangement for performing store word instruction and now we can move to the load word instruction.

(Refer Slide Time: 23:00)



Now, in the load word the address generation part is same; that we do not need to touch, this is same, same mechanism, same paths and that need not be modified. What additional thing we need to do for load word is to take the data from data memory and put it back in register file at appropriate address. So, first of all, we have this line going to wd write data of the register file (Refer Slide Time: 23:27) so we have broken this here so that we can take this and this and put a multiplexor. So there two options are and they

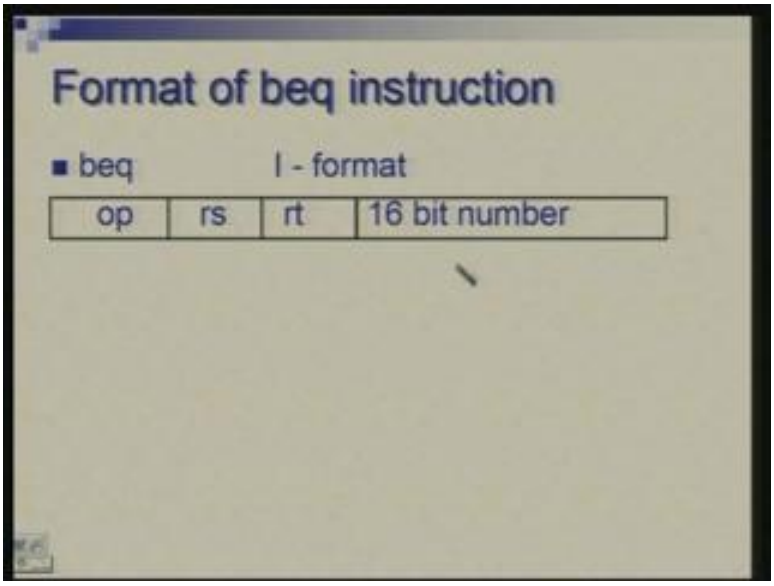
need to be joined with a multiplexor. So this multiplexor when you give control as 1 then it will send this to the register file and when the control is 0 it sends this to the register file so we will have to remember that for add, subtract, AND, OR instructions the control has to be 0 and for load instruction control has to be 1 here. So this is one part of the picture.

The second part of the picture is to give the correct write address. The write address comes from…… now let us figure out from where the write address has to come. See, for add/subtract instruction write address is coming from this part (Refer Slide Time: 24:41) bit 11 to 15 and for this it will come from bit number 16 to 20, for load instruction it is rt which is to be used here. For add subtract it is the rd which decides the right destination now it is rt which will do it. So again we need to make changes here, introduce a multiplexor because it is either bit 11 to 15 which goes there or bit number 16 to 20 which goes there so we have to make a provision for that; we remove this line and put it through a multiplexor and the two choices are either this or that. This is the choice which will be taken for load instruction, and this is the choice which is taken for add, subtract, AND, OR, slt instructions. So now we complete the……this is the complete picture for now. Seven instructions we have done: add, subtract, AND, OR, slt, and load, store. So everything which is required for these as far as data path is concerned is there.

Now let us look at branch and jump instruction. Again we will take one by one. They would need to do something with the program counter because these instructions influence how the next instruction is chosen. We did not modify this part for load store because this part continues to be same (Refer Slide Time: 26:10) you have one instruction and the next instruction follows. But for branch instruction we need to modify this and also we will use ALU for equality test. The comparison will be done and ALU will produce a bit which will indicate whether the two inputs were equal or unequal.

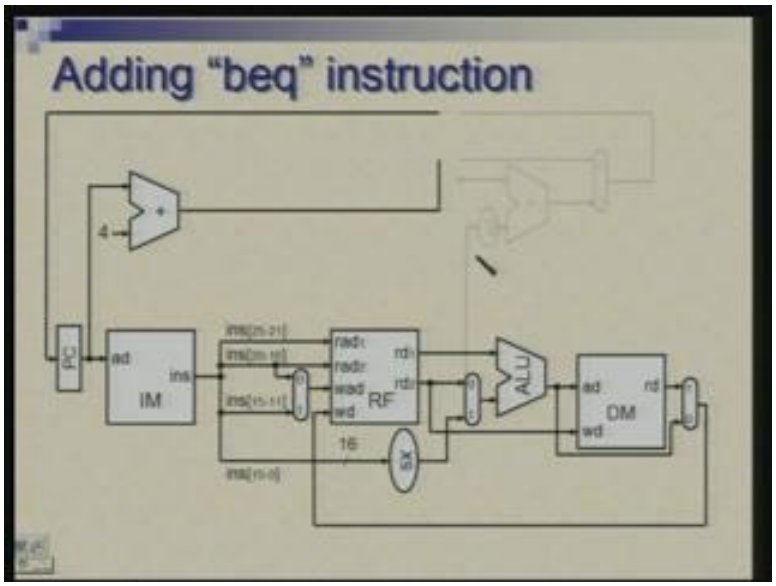
The branch instruction has this format opcode this is the I - format as we have for load store, rs and rt are the two registers which will be compared and this number would be added as word offset to PC and it will be PC plus 4 to which will be add because that part we want to retain as common for all the instructions.

(Refer Slide Time: 27:03 min)



So again start from this point where we have come up to. We need to do some modification here. Instead of sending PC plus 4 back directly to PC we will introduce more options and the options would be that this with something added to it.

(Refer Slide Time: 27:27)



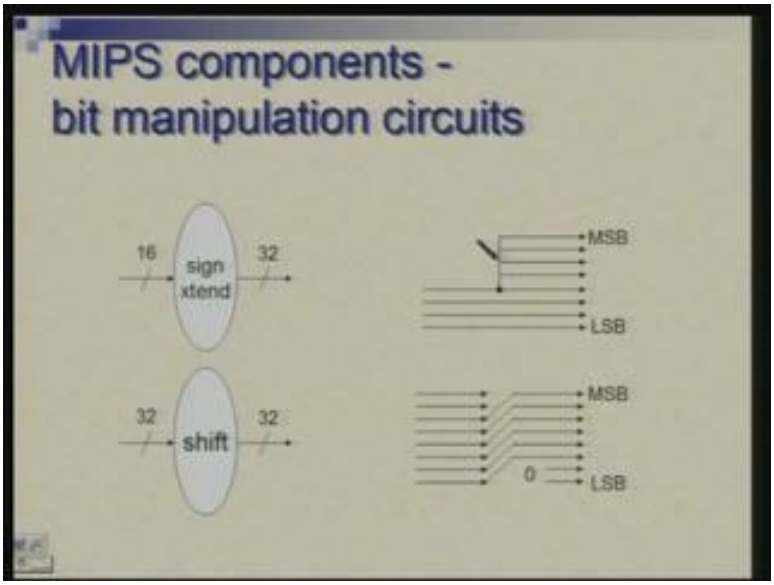
Now let us look carefully. First of all this is a multiplexor which is making a choice; either we take PC plus 4 or we take output of this adder which is adding something to PC plus 4. What is it we are adding? We are adding this 16-bit constant which has been sign extended but it is also been shifted. This s2 I am using to shift this number by 2 bits to the left which is effectively multiplying by 4 and getting a byte number from word number.

Therefore, as I had discussed earlier it is a matter of simply wiring things correctly. We have the correct offset coming here with gets added to PC plus 4 and is available to this multiplexor. now this multiplexor has to look at which instruction it is; if it is not branch instruction it will simply allow this to go through if it is branch instructions then it looks at the result of comparison in the ALU and accordingly a 0 or 1 will be chosen here. Hence, again we will get into those details when we discuss the control. But for the moment we assume that somehow there will be some logic put together which will take care that the correct value 0, 1 is given here.

So, as far as the data path is concerned what we have introduced here essentially is one multiplexor and one adder and these wiring of signals so that a shift of 2 bits takes place.

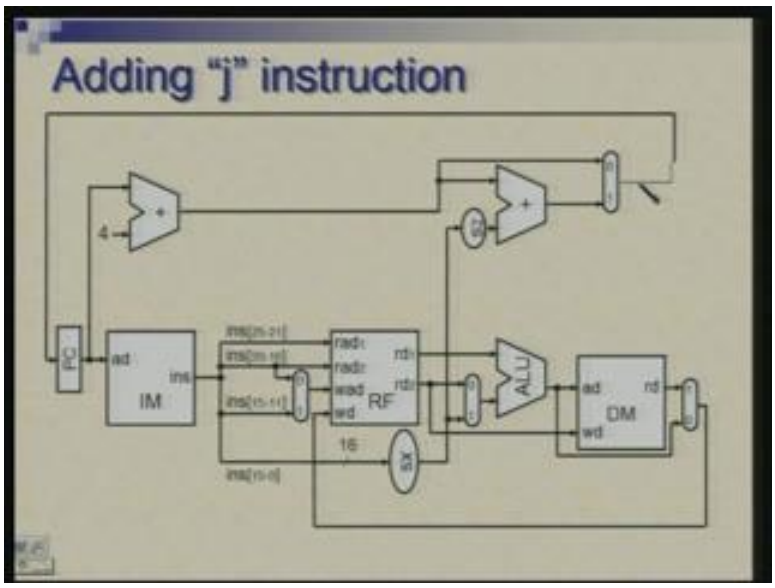
Now just to remind you we had seen that sign extension and shifting are essentially wiring there is no active gate or any active component involved. Sign extension means repeating one particular bit and shifting means just rearranging these and supplying a constant zero to some of the bits.

(Refer Slide Time: 29:22 min)



So, in the previous diagram the sign extension and the shift are essentially these kinds of wiring arrangements (Refer Slide Time: 29:42). Now finally we come into this jump instruction. Jump instruction has only two fields: the opcode field and a 26-bit field which decides the next address. Once again, since it is not a full 32-bit address we need to retain some bits of PC as it is. So what we will do is we will take these 26 bits and 4 bits from PC plus 4 again not PC and form a new address for the next instruction. So we would require another multiplexor which will provide one additional choice.

(Refer Slide Time: 30:31)

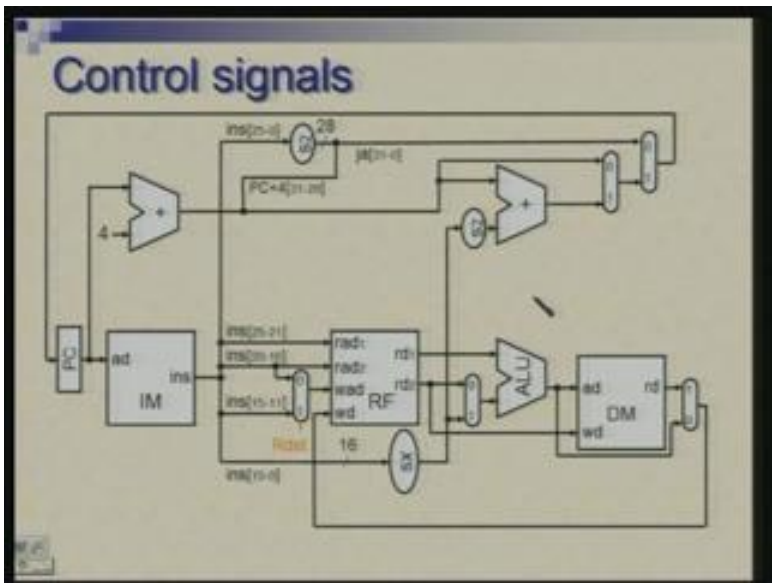


We introduce that multiplexor here, remove this line and pass through a multiplexor. And the input for this multiplexor is coming from instruction bits. We are taking these 26 bits from the instruction field, shift it left by two positions to get 28 again for the same reason we want to get byte from the word number; 4 bits we are picking from PC plus 4 bit number 28 to 31 and these 4 bits and these 28 bits are concatenated together to form a jump address which is a 32-bit value and this is available as yet another input for going back to the program counters.

The program counter PC has three possibilities: either it is plain PC plus 4 or this PC plus 4 plus offset or this combination of PC bits and instruction bits. You can actually combine these two multiplexors into a single three input multiplexor; that is another way of designing but we will just retain it in this manner.

Now, the next thing is to start worrying about how we are going to control this data path; what are the points where we need to apply control signals so that is what we will see next. All multiplexors would require control and in this exercise of identifying controls we will assign names for the purpose of reference for all the control signals so we will call this control signals for this multiplexer as Rdsd or Register File Destination. It will select the destination address in the register file; it comes from here or from here (Refer Slide Time: 32:35).

(Refer Slide Time: 32:36)



Then register file needs a control signal to tell it whether it has to write or not. All instructions, you would notice, are not writing into register file so it is only first five instructions write and load instruction; these six instructions out of the nine write into the register file so this will have to be made 0 or 1 accordingly. ALU source which I am labeling as A source is the control signal for this multiplexor and this will distinguish whether ALU is being used for address calculation or for normal arithmetic logical operations.

ALU would require 3 bits to control it. Recall the design of ALU which we have done. We had some circuit and then at the end there was a 4 input multiplexor which will select AND output, OR output or plus minus output or output for slt. So 2 bits are required to control that and another bit to choose between add subtract so there were a total of 3 bits so I am labeling that as op standing for operation. Then a status output which will come from ALU I am labeling it as z standing for 0 so it is a comparison of the two operands from the point of view beq instruction. Comparison, you recall, was done by doing subtraction and checking of the result is 0 so that is why I have labeled it as z. This is not an input, this is an output, this is the status and others are the control (Refer Slide Time: 34:30).

Data memory requires control for read or write so MR stands for memory read, MW stands for memory write. We will make sure that memory does only one operation at a time. Actually in this kind of arrangement where read and write lines are separate it is also possible that you do read and write simultaneously in this. But it has to be from the same address, same location in the memory is used for reading and writing simultaneously but in our design we do not do that; we will either do read or write.

Then there is another multiplexor here (Refer Slide Time: 35:23) which is distinguishing between what goes to register file whether it is from memory or from ALU so memory to

register file M2R. This is called P source PC source basically either here or there, then whether it is jump instruction or not so it is another control signal. Hence, these are the control signals so we require some circuit which will produce so many outputs 1 2 3 plus 3 6 7 8 9 10 11 so these 11 outputs and that control will have to look at this and also look at two fields of the instruction: the opcode field and the function field so that is the controller which needs to be designed it has to have this 12 inputs. In fact, strictly speaking, it is plus 1 so 13 inputs and 11 outputs.

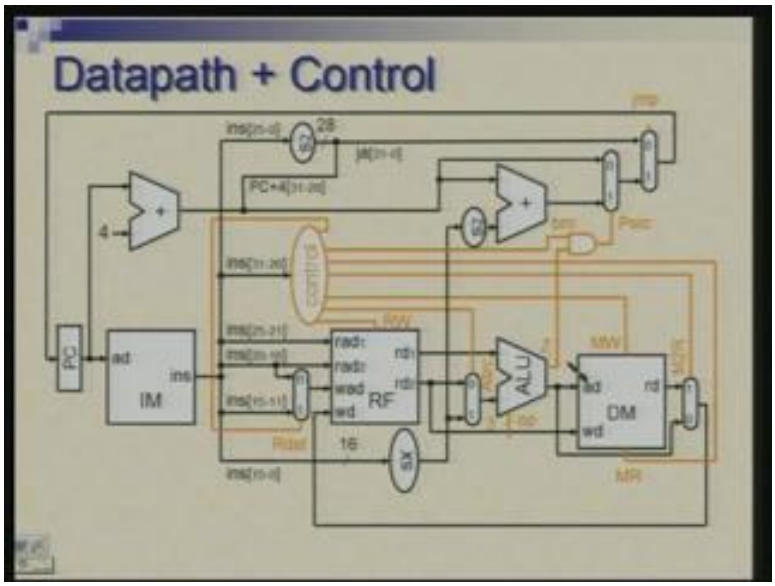
So, if you can design that and plug it in here that will complete the task. But we will not design it as a monolithic circuit; we will design it again in terms of some simpler parts as you will see now. Now you want to bring in the controller. We will not go into details but we will just go through an outline of what it is going to be.

First of all we look at the opcode bits and these bits are sufficient to distinguish between jump, branch, load, store and this group of add, subtract, etc altogether. So except for controlling ALU from these 6 bits we can derive all the information. Out of the eleven signals eight can be straightaway figured out using these. Of course generating this will require us to look at z also. So let us connect the output of this control. I will show the outputs and connect to various signals which can be driven by this.

If it is load instruction which we can figure out from here (Refer Slide Time: 37:56) we will allow this to pass through here otherwise we will allow this to pass through. So this hardness can be determined from this, whether we have to write register file or not can be determined from this, ALU source can be selected, it can be determined from this, memory write and memory read. So if it is load instruction we will do memory read, if it is store instruction we will do memory write. Then control of this multiplexor; it has to distinguish between load from other instructions so that also can be done.

Here we are doing slightly differently. We are first of all figuring out that it is a branch instruction. So this controller will activate a signal which tells us that it is a branch instruction and that we AND with this z output to control this P source. So you can see what is happening. if the instruction is not a branch instruction you will have a zero here which means that P source is 0 irrespective of what is z and the multiplexor passes the upper input. So, when it is not a branch instruction we do not branch; we do not even look at z, when it is branch instruction we will have this as 1; the controller will make this output as 1.

(Refer Slide Time: 39:45)



Now it will depend upon z; we will get 1 here or 0 here. So if the result of comparison was true if the two registers are equal this will be activated this will be 1 and you will get a 1 here so this address will go as the next address of PC next value of PC and if the test failed then this will be 0 you will have a 0 and PC plus 4 will continue.

Another way if you recall the gross diagram I had drawn where there was a data path and the controller I had shown the status signal going to the controller. So, strictly speaking, the controller is not just this, it is this plus this (Refer Slide Time: 40:28). But it actually simplifies the matter to look at it as a 6 input circuit rather than a 7 input circuit because the influence of this has been handled separately which makes it somewhat convenient.

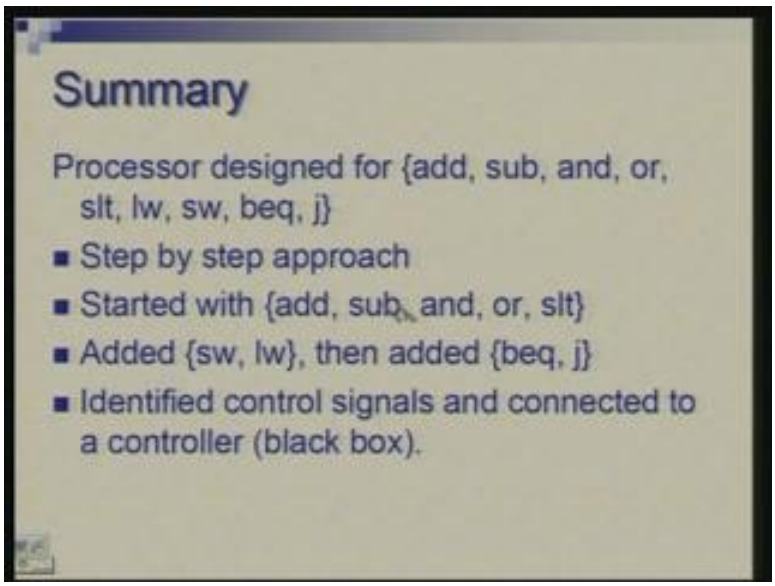
Then we can also control this multiplexor by figuring out whether it is a jump instruction or not. Now, what is left is the opcode, an operation which the ALU has to perform and it will depend upon 6 bits which come from the function field. So, the instruction 0 to 5 would be seen by another kind of control circuit I am calling it A control ALU control.

So, looking at these 6 bits we are trying to determine what these 3 bits are. But remember that ALU is being used for load store instruction also for different purposes. So we need to look at this also. But we do not need to feed all those 6 bits here but what we need to tell this circuit is which class of instruction it is; whether it is R - class one of those instructions or it is load store or something else. So we will have some information coming out from here connecting to this and this needs to be just a 2 bit information because we like to distinguish between three different cases: the R class instruction, add, subtract, AND, OR and slt that is one possibility; load store together grouped as another possibility and all the rest as third possibility. So we will take 2 bits out of this and connect to this and I am calling it opc which stands for operation class.

So basically this circuit now has 6 inputs plus 2 inputs so 8 inputs producing 3 outputs. So we have taken special care to keep the number of inputs low so the complexity of a combinational circuit like this or like that depends upon the number of input lines and number of output lines. But I will not explain that but you can take it that with the number of input lines the size of the circuit or the complexity grows generally exponentially. But with outputs it will generally grow linearly. So we are more worried about keeping the numbers of inputs as low as possible. So, if there are large numbers of inputs then we can look at the parts of it separately and then combine the results that are always a better strategy; that is what we have followed.

This circuit is predominantly looking at the opcode field (Refer Slide Time: 43:38) whereas this is predominantly looking at the function field and some additional information it requires about opcode is being actually processed by this and made available here. So, after having done this what remains is basically to look at the design of this and design of this. So if you can enumerate what outputs you require for what inputs you can do it in a straightforward manner and same thing here. We will take it up in the next class.

(Refer Slide Time: 44:13 min)



So just to summarize, we have designed a processor except for some controller details for these nine instructions add, subtract, AND, OR, slt, load word, store word, beq and j and our approach was a very simple step by step approach; we gradually put the components on the table, connect wires and build up the circuits. So first we took this group of first five instructions which are similar in nature the only difference comes in the way you control the ALU so we did that we started with these five added, store and load and then added, beq and jump; then we identified the control signals for the various components which were put there and we placed some black boxes which are called controllers which expected to produce these signals and the next task will be to look at the details of these two black boxes and design this. That is all for today, thank you.