**Computer Architecture**
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture -21
**Processor Design - Control for Multi Cycle Design**

We have been discussing design of a processor where execution of each instruction is divided into multiple clock cycles. We have seen how the data path is designed and today we will look at the controller aspects how such a data path could be controlled.

(Refer Slide Time: 00:01:20)
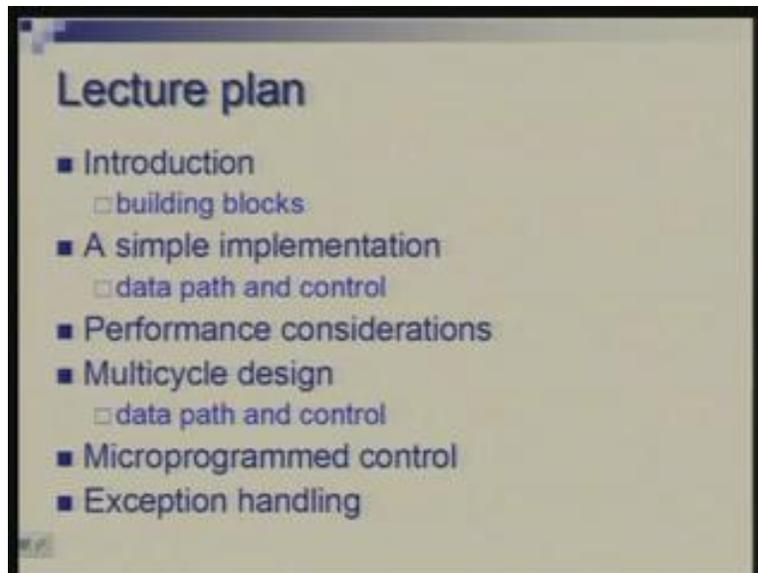
In the sequence of lectures on designing of processors we started with very simple design where everything about an instruction was done in a single clock cycle. We notice the problems with performance and there were other issues so we have moved to a different style where the instruction is divided into multiple clock cycles. So we will look at what are the actions which are done in different clock cycles for various instructions and then by putting these sequences of actions together we will try to build the flow of control for carrying out these instructions.

(Refer Slide Time: 00:02:44)
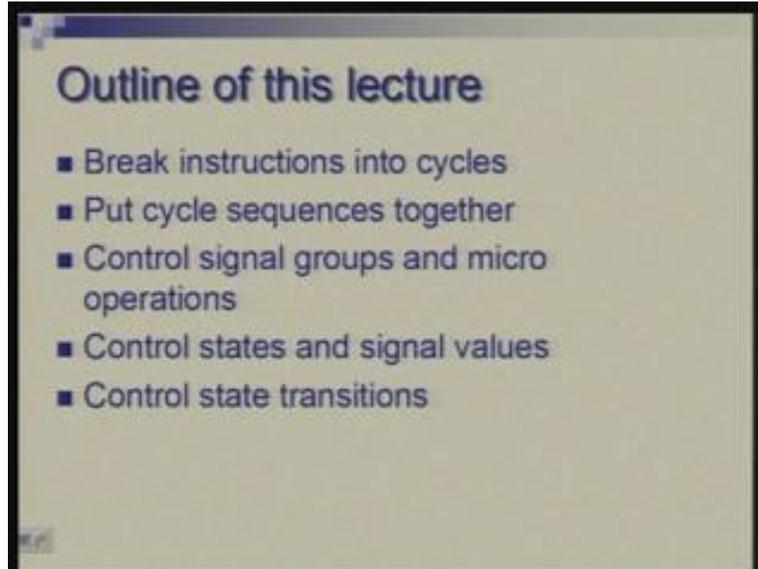
We will then try to identify what control signals are required to control this data path in each of the control steps; for doing so we will group the control signals into groups and we will define some meaningful operations called micro operations. So each instruction would be viewed as a set of micro operations which are done either together or in sequence and that will somewhat simplify establishing a relationship between control states and the signal values. Finally we will see how control states transit from one to other and that will complete the design of control part.

(Refer Slide Time: 00:03:00)
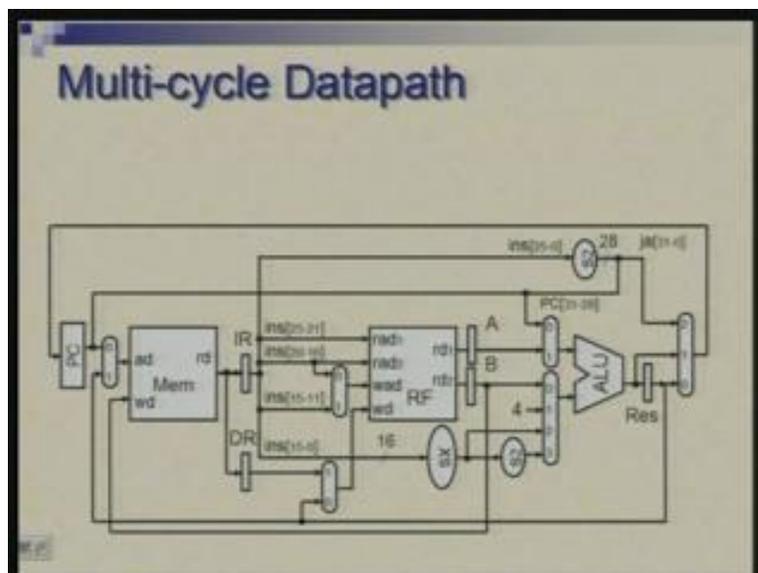
So this is the starting point, we had arrived at this data path last time where the key resources are memory, register file and ALU and we have tried to use these as far as possible so that there is a maximum utilization of all the resources and you would also recall that all intermediate results we try to store in some registers. So, for example, when instruction is brought out from memory it is stored in instruction register and when data is read out from memory it is stored in data register. Similarly, operand from register file is brought out into registers A and B and the results of ALU operation are kept in register called Res short for result.

So now we need to control all these components; all the registers may not change their states in every cycle. So each register will have a signal indicating when does we write something into those registers. We have controls for the multiplexor as usual and similarly controls for ALU, register file and memory. But since we have new components like registers, multiplexors also have either changed in their size or in their organization we will have to redefine some of these control signals.

(Refer Slide Time: 00:04:36)

Break Instruction Execution
into Cycles: R-class instructions
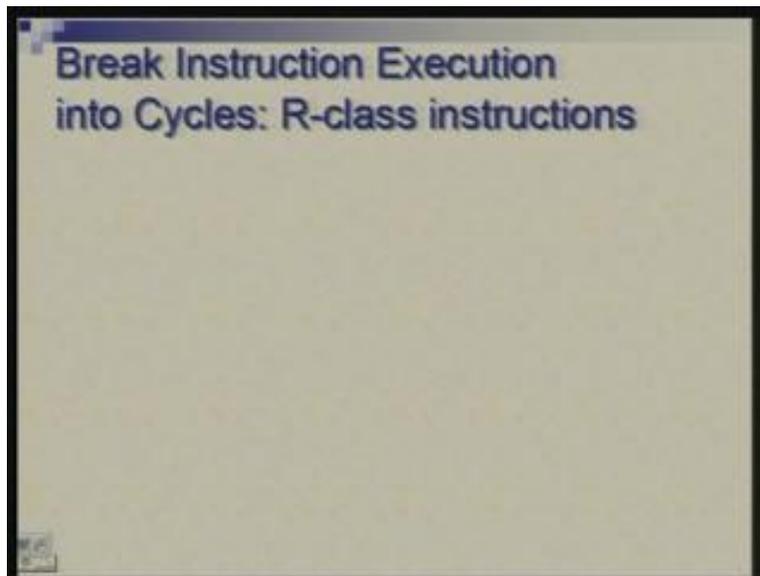
So, first of all let us get back to the instruction and see how those instructions are divided into operations which were done in different cycles. So what activity is done in which cycle that needs to be clearly recorded before we can start working with the control signals?

(Refer Slide Time: 00:05:04)
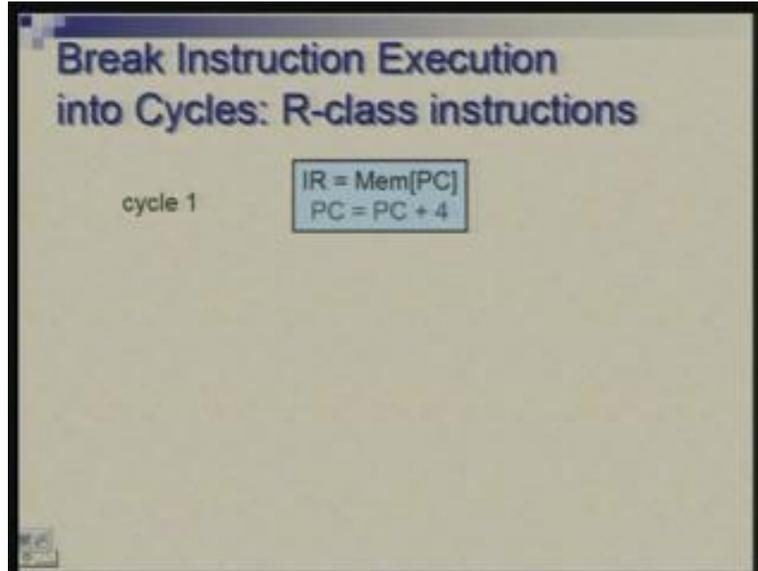
So, starting with R class instruction in the first cycle we read one word from memory into instruction register that forms the instruction, address of memory that comes from PC and at the same time PC takes on the new value. So both these operations are done concurrently within first cycle. In the next cycle we read the operand from the register file and these two are brought into A and B. the addresses of register file are provided by the instruction and the relevant fields here are bit 21 to 25 and bit 16 to 20 for the second operand. So this corresponds to…. the first one corresponds to RS, the second one corresponds to RT.

Once again what we meant by putting these two operations in a single box was that both of these are done simultaneously within a clock cycle number 2.

The next clock cycle for these instructions would see the actual operation being performed by the ALU. I have written in a generic sense A op B where op is the operation as would be guided by the function field of the instruction IR (0 to 5) (Refer Slide Time: 6:37) and the last cycle will involve transferring this result to the register file; the address come from bits 11 to 15 which corresponds to RD or the destination register. This is how things have been divided and we have made a careful choice of what gets done concurrently and what gets done in sequence.

(Refer Slide Time: 00:07:01)
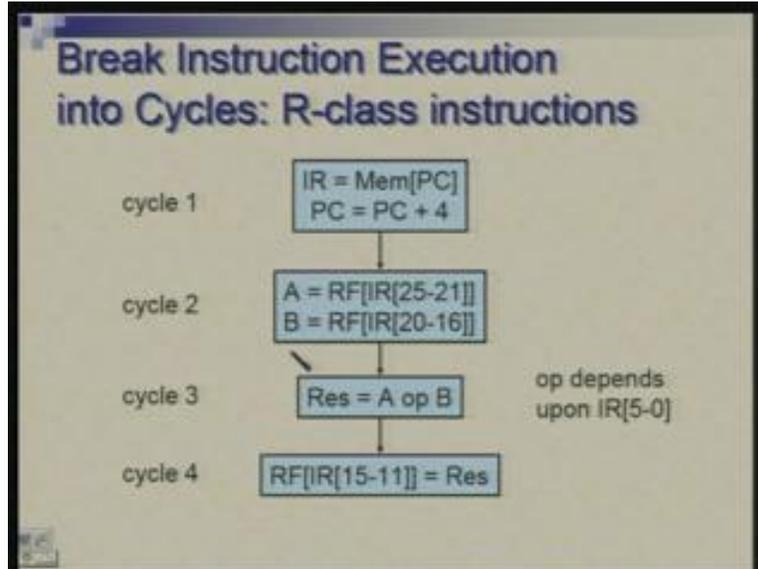
The second instruction we will take is store word and the first cycle involves access to memory to fetch the instruction and updation of program counter. The second cycle involves bringing the registers out we need again to access two registers: one which will participate in address calculation and second will carry the value to be written into the memory.
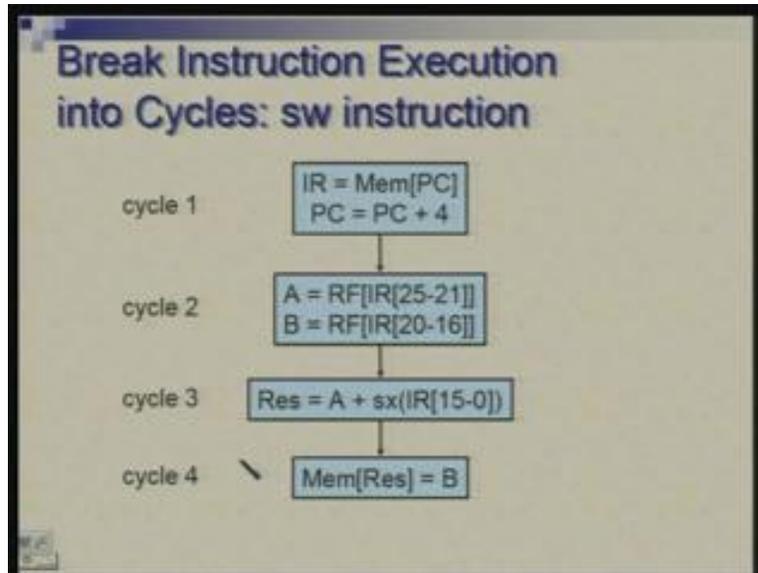
(Refer Slide Time: 7:35)
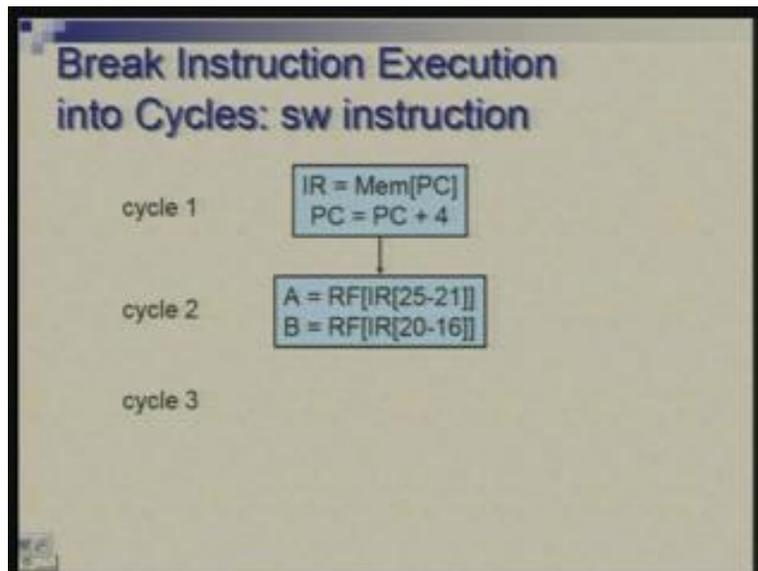
So with same fields of the register file the same fields of instruction register file is accessed and values are brought out in A and B. In the next cycle we calculate the

address (Refer Slide Time: 7:53) by adding offset coming from bit 0 to 15 of instruction with sign extension to A and this value is temporarily kept in this register called result.
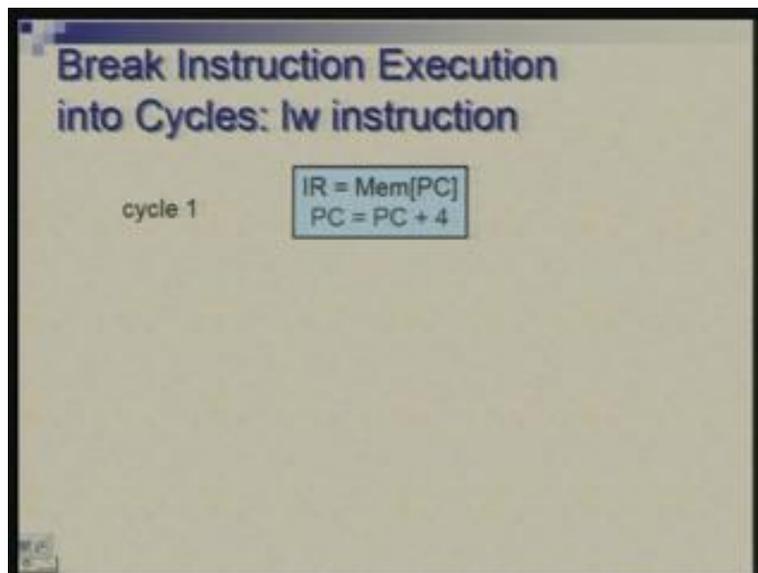
The last cycle will see an access to memory where contents of Res will be used as an address and the data to be written into memory is B and memory write is performed. So once again we require four cycles.
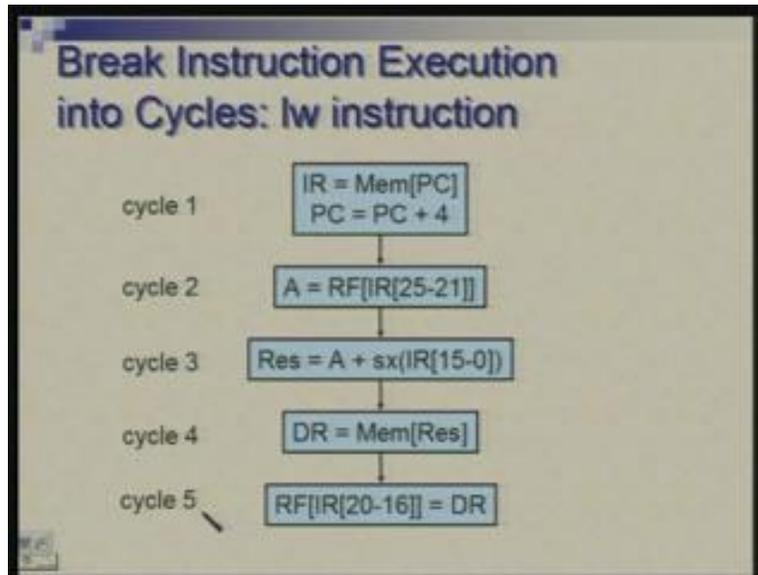
(Refer Slide Time: 8:23)

The next is load word instruction.

(Refer Slide Time: 8:30)

The first cycle again is similar. The second cycle involves reading one register from register file. Here we need to access only A because the second address corresponds to the destination here. What we read from memory will be stored in register RT so we need to read only one register and in the third cycle we calculate address in the same manner. In the fourth cycle memory access is performed; the data is read from memory, address is carried in Res and the data which is read is brought into DR. In the last cycle we store the DR value into register file, the address comes from RT. So this is the sequence of five cycles which complete load word instruction.

(Refer Slide Time: 9:30)

(Refer Slide Time: 00:9:36)
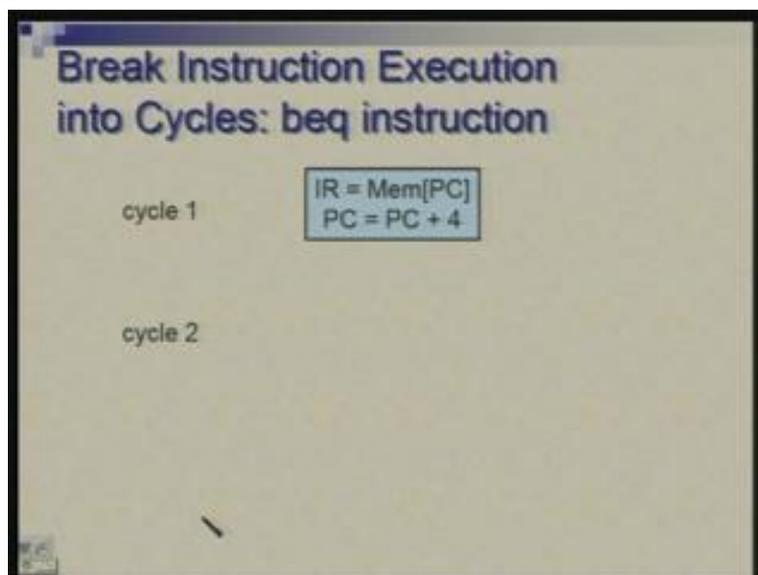
Then we move on to beq; first cycle is same. in the next cycle we are doing lot of work. one is to access A and B the two operands which need to be compared and we also keep the result ready for the target address in case branch has to be carried out because the ALU is free in this cycle. In the next cycle we are going to use ALU for comparison. So in this cycle it is free. We calculate this address but do not transfer it to PC immediately. We keep it in Res; the condition is yet to be checked. So, after checking the condition we will either transfer it or not transfer it. That is cycle three where the ALU will compare A and B, the two operands which were brought out from register file and if the condition holds then this new address is transferred to PC. So this requires only three steps.

(Refer Slide Time: 10:40)
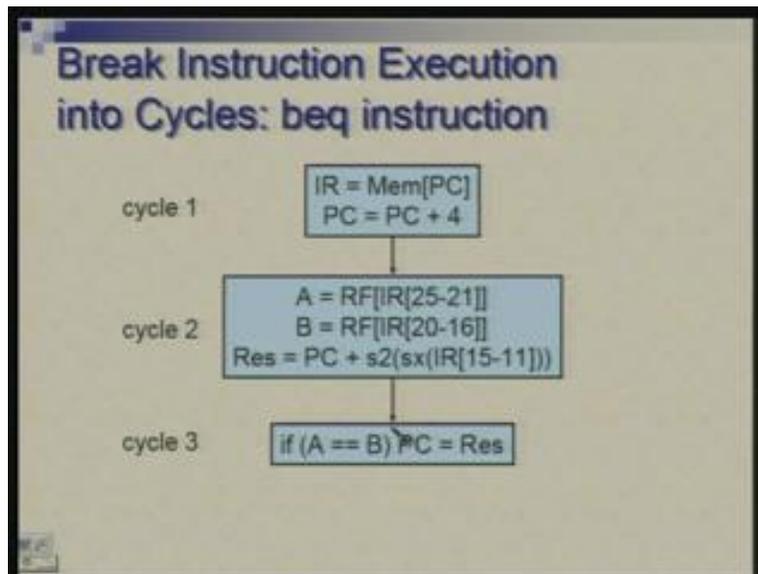
Lastly we will look at the jump instruction. again, the first cycle is same and in the second cycle we compose this address of the next instruction by taking bits from PC and IR with I missed out…. oh no, there is no sign extension here; this with 2 bit shift is concatenated with 4 bits of PC and then transferred to PC. So this is done in two cycles. Now you recall that the impression I might have given earlier was that this can be done in a single cycle because only resource which is time consuming which is required here is accessing the instruction after that there is no memory access required, no ALU operation required so the total delay which we always use to think for this was max of t i and t plus.

(Refer Slide Time: 11:48)
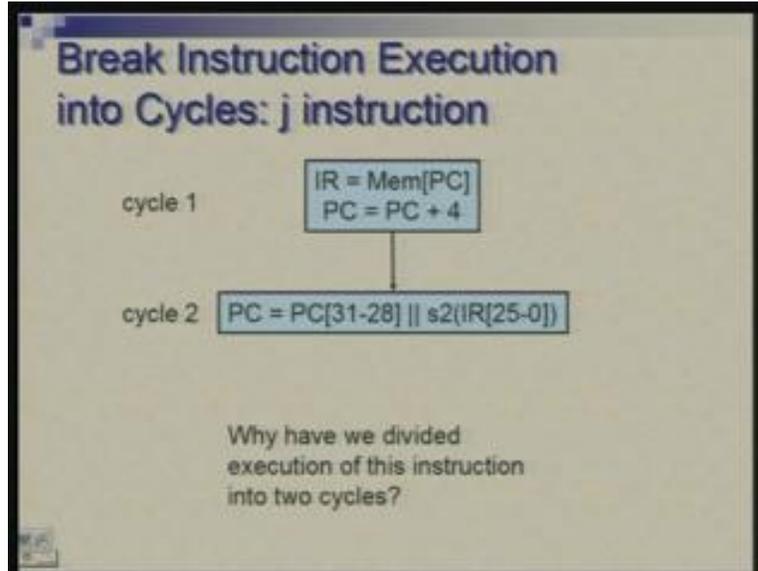
But why are we occupying two cycles here. The reason for that is we want to do this operation sequentially. The 4 bits we need to pick up from PC part is actually after PC has been incremented. So, after PC is incremented one cycle is over. In the first cycle we are incrementing PC and the result is put back in PC. So those 4 bits are picked up in the next cycle and put together with the instruction. Also you would notice that it takes one cycle to fetch the instruction. So although what is being done here is not a time consuming activity but since it has to be sequenced after that it is occupying additional cycle. So in a in a single cycle approach what was roughly taking equivalent of the memory access time or addition time one of the two now it is taking more or less twice of that.
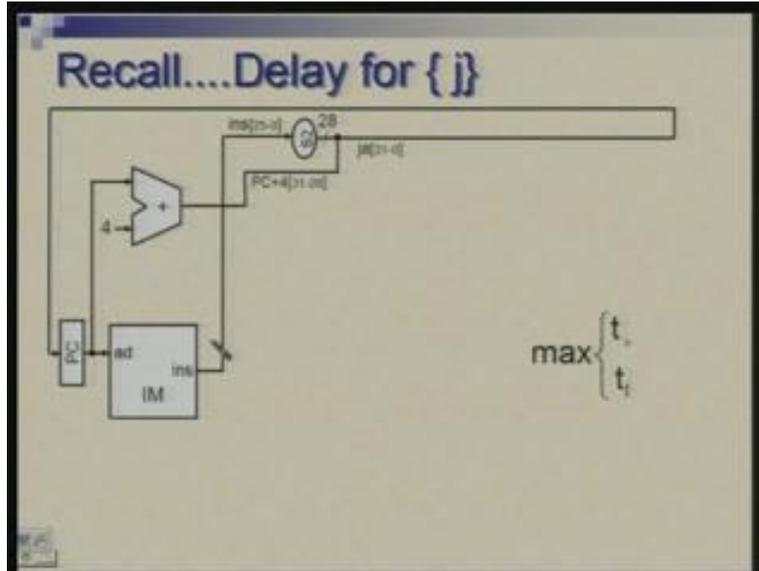
This is just to recall that in the single cycle approach the data path was like this that we did instruction access and PC update and directly we picked up the bits and formed the address and we said that the time required is max of t plus and t i. So now if we have decided one clock cycle to be let us say something which encompasses these we are still requiring two cycles because of the need of sequencing things.
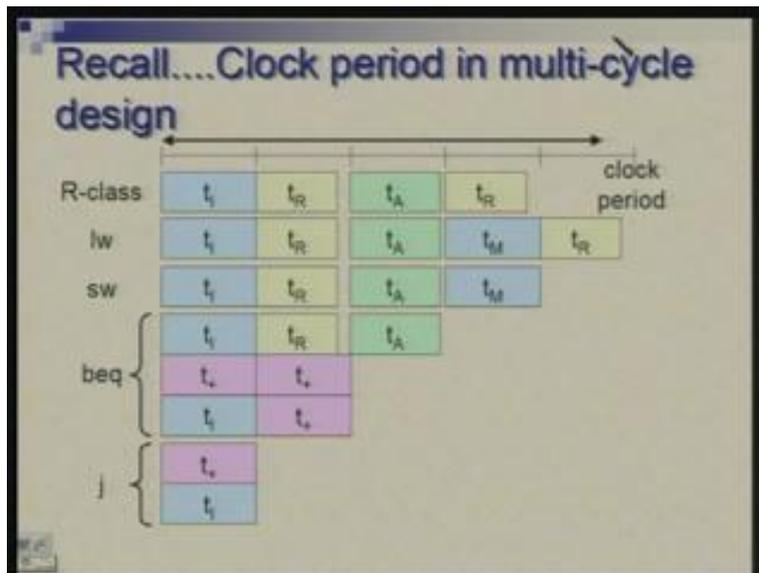
These were the timings we had conceived earlier and we were actually imagining the time for jump to be just this much. basically first clock period which will accommodate t plus and t i would be used but we are doing something now; a very simple activity but

now since the time is quantized discretized in terms of the clock period we need to go beyond this and we cannot use anything less than a cycle. We use two cycles for this instruction.

Actually as we will proceed further we will change things even for little worse but that becomes essential. Now, we have seen the division of each instruction into cycles; the division of activity of each instruction into cycles separately and once again we need to put things together to form overall flow of control. So these are the five instructions or groups and we will put their actions one after another in the same picture so that we have a global view of the whole thing.

(Refer Slide Time: 00:15:04)
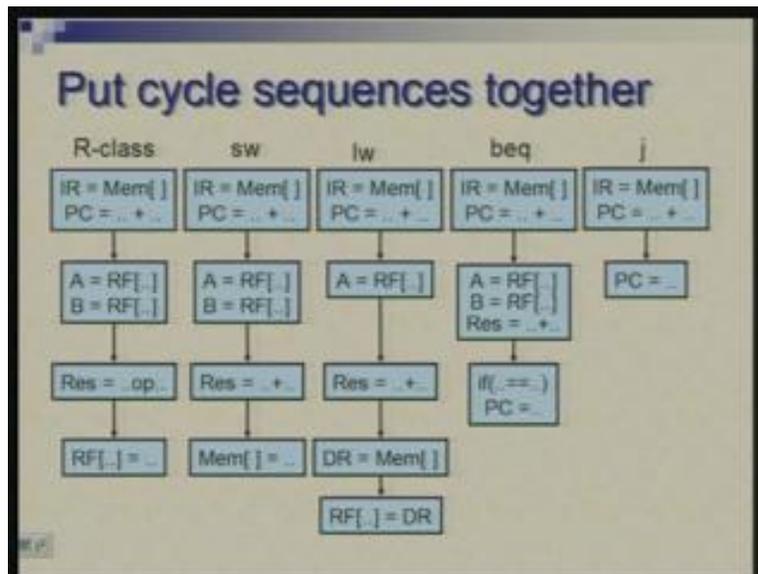
Now I have just, to accommodate everything on a single screen, I have omitted some pieces of text which are more a matter of detail but what I have tried to retain is all the destinations of the operation wherever the results are going and the main resources which are being used. So, for the first cycle in all cases you would see that there is access to memory I have omitted some details here, there is updation of PC, excess of values in the register file performing this arithmetic or logical operations, storing in register file, storing in memory, reading from memory and so on. So you can see that the essence of all these the skeleton of all the operations has been captured here.

Now the task is to put these together. We have different instructions taking different cycles. You would notice some commonality something is common and that is where we can merge but in the flow of control at some point we would need to branch because different instructions require different actions. So the first part is; first cycle is apparently common to all of them. So we could start always with that action and a common state for all the instructions and once things start differing we branch to different states. Different boxes will correspond to different state in which the controller would be and given a state of the controller we would know clearly what actions are to be carried out in that

particular state. So what we are trying to arrive at is some sort of state transition diagram which would describe the control.

At the moment we have five different chains or five different sequences but we want to have a single graphs which indicates how one moves from one state to other state and what action is required in each state. So, obviously these can be merged. The first cycle seems to be carrying out same activity and we can merge this.

(Refer Slide Time: 17:24)
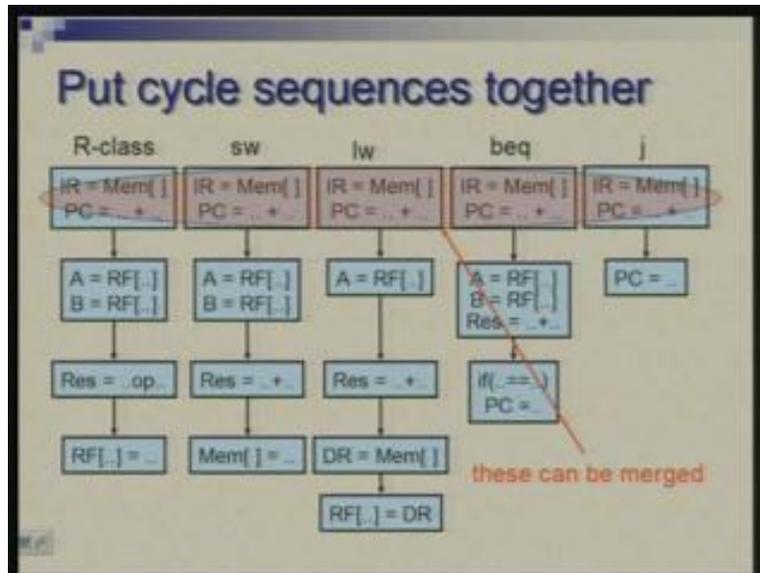
Therefore, after merging we have a single state and then a point where we are branching.

(Refer Slide Time: 17:46)
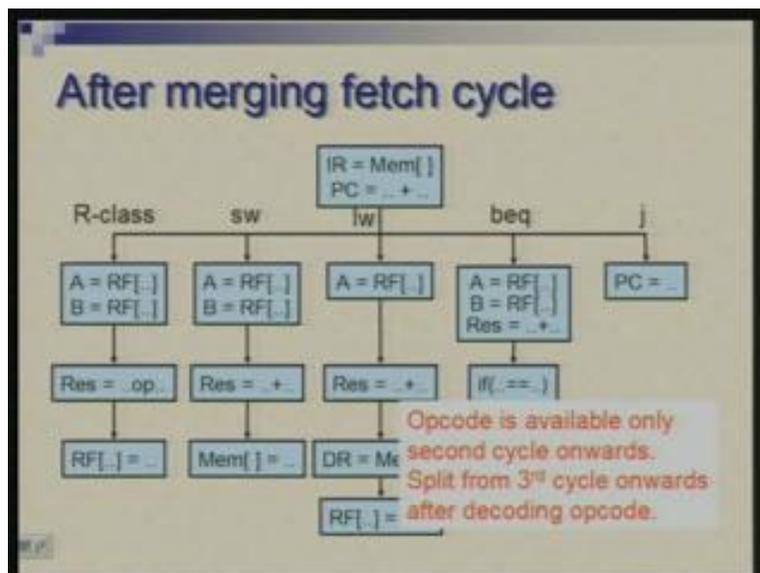
Now, another important thing which needs to be noticed at this stage is we are contemplating a bifurcation here; we are splitting from this point onwards after first but this requires us to look at the opcode. Before we can split we need to see what the opcode is and which instruction or instruction group it is. But you can start looking at opcode only after you have fetched the instruction and brought into IR. So now this is available, therefore instruction is available to you from second cycle onwards and therefore you can start looking at opcode from the second cycle onwards.

Typically in a complex design this decoding of opcode or understanding which instruction it is could also be a time consuming activity. The circuit which is going to identify a particular instruction would typically be allowed a full cycle to do it. Of course we do not need that but still the opcode decoding will have to take place in the second cycle although we may not have the second cycle fully occupied. Therefore practically this bifurcation or split must occur after second cycle because if you have to go from this state to one of these states we should know what the instruction is here but instruction is known only after this first step is completed. So it is only after second step we can branch off to different chains and do special action necessary for various instructions. Therefore second step will also have to be common.

Now let us see what is required to make it common. One thing is that not all instructions are trying to read two values from register file. Here you need two values, two values; one: here you are reading two values, here you are reading none. Is there any harm if all instructions read two values; they may use it or may not use it. The answer is there is no harm; it does not cause any problem in the functionality of the instruction; you might consume some energy in doing so but let us keep that aside and agree to fetch both the values in all the instructions in an attempt to come up with a common action for the second cycle.

The other thing you notice here are that address is being calculated here which may be useful for branch instruction and this value is kept in register called result. So what if we repeat this also in all the instructions. Once again ALU is free we are not doing anything with it so if we occupy ALU in an activity which we may discard later on still there is no harm apart from energy consumption. Also, the result register is not holding any value which will get overwritten here. So what we will do is let us try to make this as the common action for all the instructions; the only trouble comes with jump here (Refer Slide Time: 21:14) because jump requires that we transfer a new value to PC but that we cannot do for all the instructions; that would mean that after every instruction a jump will be carried out. Therefore we need to postpone this to the third cycle. We will do a common action in the second cycle which is same as the beq action. For all other instructions there may be some superfluous or unnecessary activity which may get discarded but there is no harm. The only harm which is occurring is that this action PC getting a new value is getting postponed.

(Refer Slide Time: 00:22:10)
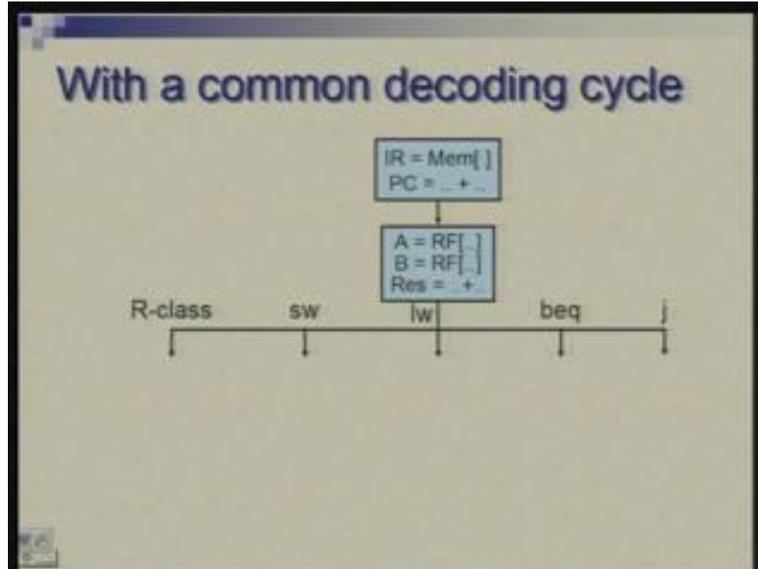
This is the picture with a common decoding cycle. This cycle is actually often referred to as decoding cycle or operand fetch cycle. Now, this jump instruction has taken another hit here and we are eventually using three cycles for it. But please remember that this is an instruction whose frequency of occurrence is comparatively much much lower than others and therefore the overall impact of this loss in the performance will be negligible so we do not really mind it. And now we have a very clean situation that there are two common cycles. By the end first cycle we know what instruction it is and we are ready for the next value of PC. In the second cycle, after the end of second cycle (Refer Slide Time: 23:09) our operands, if we need, are ready in A and B and the branch address is ready in Res if we need later on and then we can move on to one of these separate branches. So, two further cycles are required for R class, two for store word, three for load word, one for beq and one for jump and with that all the instructions will be over.

(Refer Slide Time: 23:35)
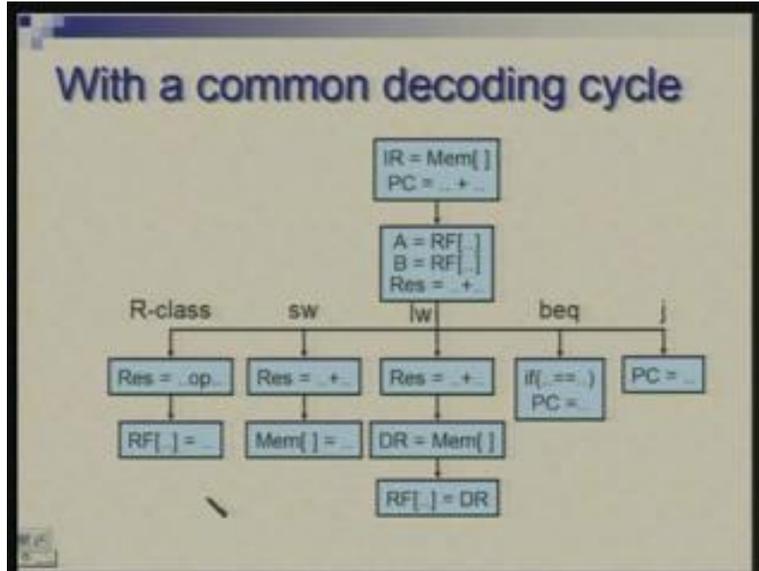
So now this is the cycle, this is a broader cycle (Refer Slide Time: 23:43) which needs to be repeated over and over again. So therefore, after these last states in the chain we come back to this. Now, as far as controller is concerned it is a small finite state machine with as you can see these about ten or so states and it keeps on cycling through these. So here you do the fetch, decode, (Refer Slide Time: 24:07) you come to know what instruction is, follow one of these paths and then you are ready for the next instruction. So, as long as power is on on the processor it will keep on going through this overall thing….. so this whole thing is called instruction cycle and within this you have clock cycle.

(Refer Slide Time: 24:25)
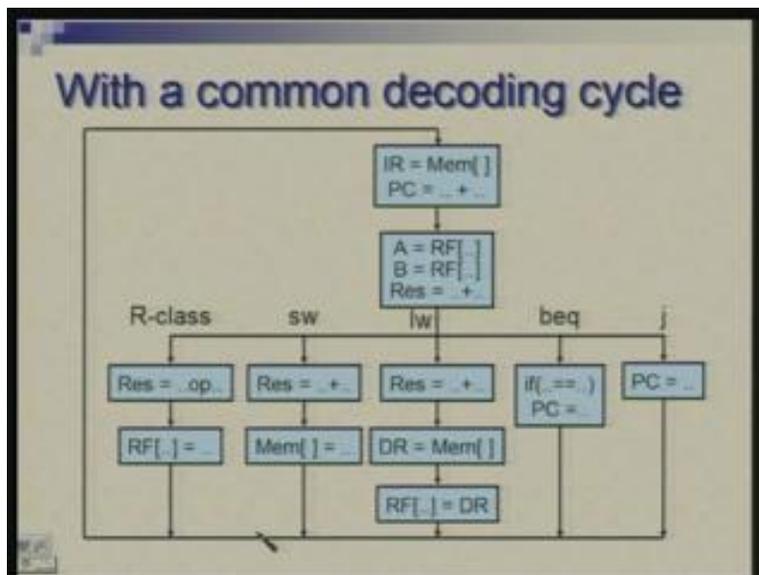
So an instruction cycle would require four clock cycles or five clock cycles or three clock cycles as the case is that is in this particular design but there are of course machines where this range of number of clock cycles could be much much varied.

Now we need to worry about what action we perform in each of the states. So these are the states (Refer Slide Time: 24:56) but before that there is another small improvement possibility here which we will notice is that load word and store word have one more cycle in common where the address is being calculated and we can actually merge that and as far as load store are concerned we keep them together up to the third cycle and then bifurcate into load and store so that reduces number of control states by one more.

(Refer Slide Time: 25:31)
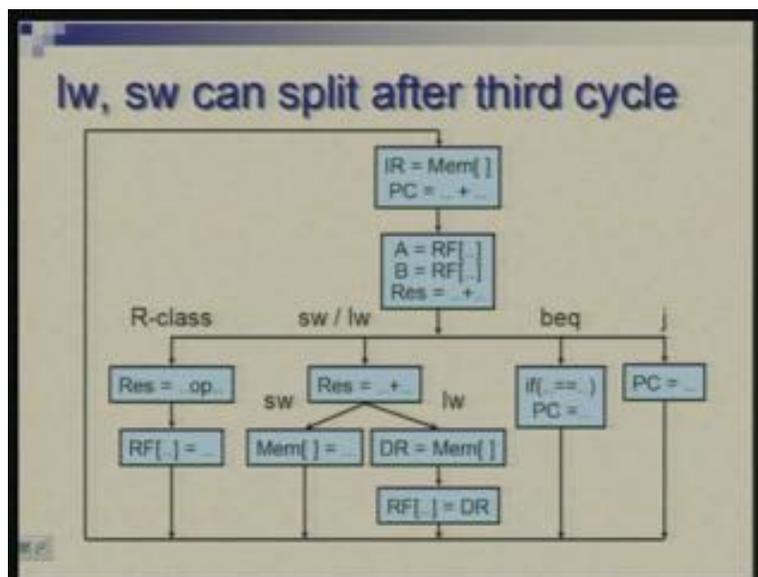
So total now I have how many states 1 2 3 4 5 6 7 8 9 and 10 now there are total of ten states and we will proceed further by looking at now the control signals.

(Refer Slide Time: 00:25:46)
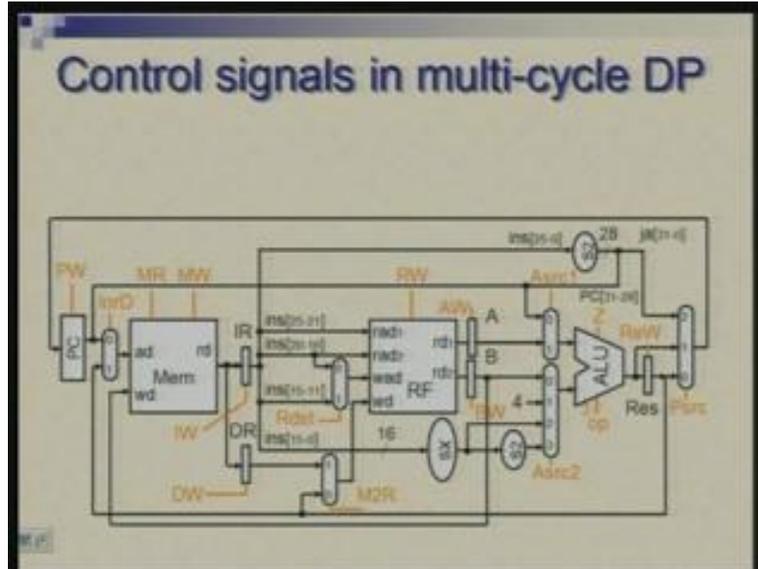
So this is back to the same data path. We have now roughly seen how we need to exercise these and what needs to be done for various instructions in different cycles; so that picture has been made very clear now. We get back to each of the components which require a control and try to understand how we need to control it. So, for each of the register I have indicated a control signal. So, for PC write I am abbreviating as pw. This is the IR write IW, DR write DW and then A and B have their AW and BW signals and Res write is ReW. So there are signals, control signals which will indicate whether a particular register changes its state in a cycle or not and in which cycle the state is changed we will know from this flow chart. We will always refer to, now, this flow chart or this state transition diagram (Refer Slide Time: 26:54) and then decide the values.

Now we have also 1 2 3 4 5 and 6 multiplexors each one requires some controls so control for this multiplexor decides whether we are accessing instruction or data and accordingly the address comes from different sources. This multiplexor decides whether we are writing into rt or rd, this decides what gets written into register file (Refer Slide Time: 27:28) we are calling it M2R memory to register or it is ALU out to the register. The name of this signal is same Rdst we had earlier. this multiplexor is controlled by a signal called A source 1, this is controlled by A source 2 and we had earlier two registers….two multiplexors which were handling the next PC value; now it is the single multiplexor with three inputs and the control signal is labeled as p source or PC source. The memory register file and ALU have their usual control signals same as what we had in the single cycle data path.

Now you would notice that there are many more control signals as compared to the single cycle data path we had. So I will not try to build the table exhaustively for all of these; what we will do is we will group the related control signals together and also identify the meaningful operation which we call as micro operations. For example, PC plus 4 going to PC will be considered as a micro operation; it is an understandable action within itself

and it will affect some of the control signals in the data path. So we will be grouping the signals according to our logical needs and then try to look at things group-wise so that will simplify the matter substantially.

So first we talk of a group of signals called PC group which are related to PC; program counter and its address. So I will build a table where I will list micro operations which are related to PC and the signals which are related to PC. So the signal you can see right now; one is Psrc the last multiplexor which is being controlled by this and the write signal for PC I have split into I have used two signals PWu and PWc; PC write unconditional and PC write conditional. you would notice that in some micro operations like PC gets PC plus 4 we are writing it unconditionally whereas there was an operation like this (Refer Slide Time: 30:22) where we are writing into PC with some condition. So this state will generate a signal which I am calling as PWc conditional and a state like this or like this will generate a signal which I am going to call PWu unconditionally. So the signal PW which is going here would be derived out of these two; I will explain that in a moment. But let us see different micro operations and the signals which they imply.

So, for doing PC gets PC plus 4 I make PWu as 1, this then I dont need to care and the source I am selecting is 1 so three things are going into this multiplexor if you recall is output of ALU directly, output of ALU through Res register then the address which is for the jump instruction. So let me just go back and check if I have indicated the correct source here.

(Refer Slide Time: 31:29)
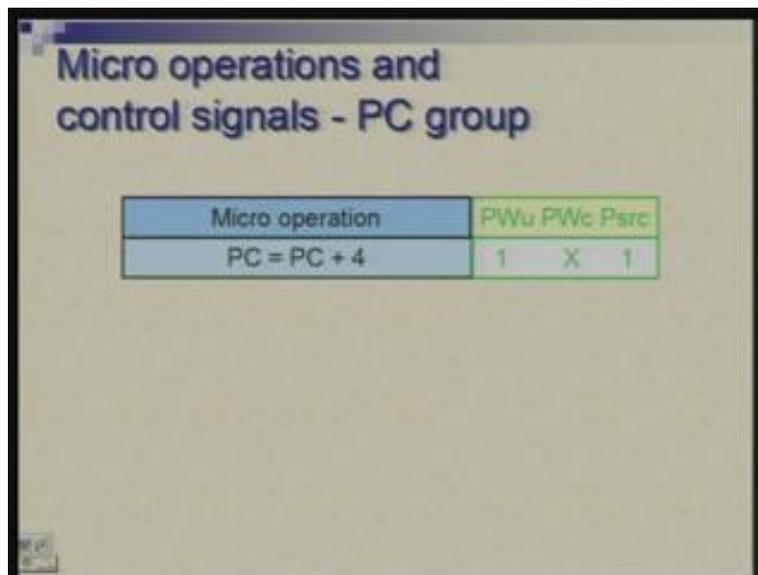
So these are three inputs to this multiplexor (Refer Slide Time: 31:38): after the register, before the register and this jump address. So recall that we will write output of ALU directly into PC when we are doing PC plus 4 so we do not bring register into picture. But the target address for a branch we are temporarily keeping into this we are not

directly transferring to PC and therefore when we transfer to PC we will take from output of the register therefore both paths have been provided.

One more thing which we should see here is the way z is going to be used. In the single cycle design we used z directly to control a multiplexor through AND gate so there was a signal coming from controller that was ANDed with z and we controlled a multiplexor. So basically we were making a choice between PC plus 4 and PC plus 4 plus offset one of the two going to the PC. But now things are handled little differently because PC plus 4 is sent to PC unconditionally in the first cycle.

In the third cycle the choice is either to transfer the new address or not to do anything. Therefore, effect of z will be brought into the way PW is being generated. So, in the first cycle we will have output of ALU directly through this multiplexor going into this, in the second cycle the address is calculated and kept in Res and in the third cycle we will bring this out here without looking at z but we will look at z and decide whether to transfer it or not. So you will see that z will come into picture when I define how PW is derived from PWu and PWc.

(Refer Slide Time: 33:47)
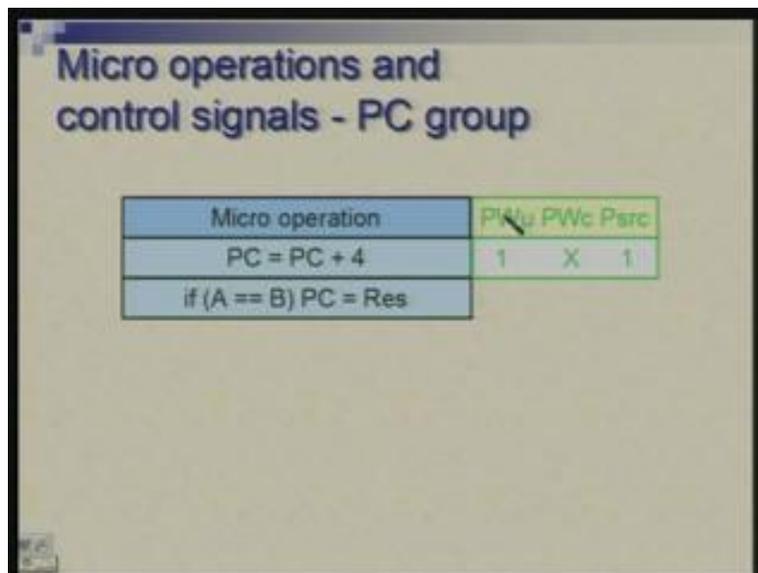
This is next micro operation where we conditionally transfer an address to PC. Here I activate this signal and do not activate this signal the source is 0 which means this value which is in Res is being taken and the third case is that the jump address goes to the PC. So again this is unconditional, so PWu is 1, PWc is x and source is 2. So in these tables I am going to write 0 1 2 3 etc when the signal takes multiple values more than two values. But actually you can think of the binary code of these going to the components and not this decimal value.

So, apart from these activities I would also like to define what is the default value of these signals. When I am not doing any of these what should I feed to these particular

points so the default value should keep everything inactive. Therefore, both write signals are 0 and then of course once that is the case the source does not matter so I put an x and here is how PW is derived from Pwu and PWc. So, if PWu is 1 PW becomes 1 irrespective of what is that we have here. But when PWu is 0 and PWc is 1 then it is Z which dictates what we get here. If both are 0 then again Z gets ignored. So basically I need one AND gate and one OR gate so the controller will produce these two signals and with the two gates I will derive PW which gets connected to the PC.

(Refer Slide Time: 35:51)

This is the address which is being formed for jump instruction (Refer Slide Time: 36:00) and in one of those cycles…… right now we are not worrying in which cycle what is happening but all we are worrying is that given an action like this to be performed how do we control the data path. So, to make this happen we need to make the unconditional write signal as 1, we do not care what PWc is because you see when PWu is 1 the values here do not matter the result here is going to be 1 it is a sort of overriding signal. But on the other hand, when PWc when we want to activate PWc we have to make sure that this is 0 because this will otherwise suppress that.

So PWc is don't care and we need to select the correct value to go to PC. So, in all these operations, notice that PC is the destination but the sources are different. There are three different sources; and the multiplexor is selecting the sources (Refer Slide Time: 37:07). This is 0 sorry this corresponds to 0 input of multiplexer, 1 input of multiplexer and 2 input of multiplexer this is all the things that have been connected in the data path so that takes care of what the value is being transferred and these two signals are taking care of whether this transfer is taking place conditionally or unconditionally and that default.

Hence, there are many control states where we are not changing the value of PC and there we need to keep both these as 0 and the value of P source does not matter. So, I would give some names to these which will make things convenient in subsequent discussions.

The first one I will call as PC increment, this is branch and this is jump and this I call no op or no operation. So just some names for convenience I have assigned to these.

(Refer Slide Time: 38:10)

Micro operations and control signals - PC group

Micro operation		PWu	PWc	Psrc
PC = PC + 4	PCinc	X		1
if (A == B) PC = Res	branch	1		0
PC = PC[31-28] || s2(IR[25	jump	X		2
default	nop	0		X

$$PW = PWu + Z \cdot PWc$$

Now let us look at operations which revolve around the memory.

(Refer Slide Time: 38:20)

Micro operations and control signals - Mem group

Micro operation	MW MR IorD IW DW

The relevant signals here are memory write and read control signals, this I or D instruction or data this signal decides what is the source of address for the memory and these two signals decide where we keep the things which is coming out of the memory when we are reading from memory. So, this controls writing into IR register, this controls

writing into DR register and one operation is fetching the instruction. So we are not writing here we are reading; it is an instruction and the multiplexor code for that is 0, the value which is being read is written in IR register so we keep this 1 keep that 0; next is getting data from memory so again we are reading we are not writing, it is data so that makes it 1, we are not storing into IR we are storing into DR so that is 1 and that is 0; next is writing into memory so now we make write as 1 and read as 0 once again it is data so I or D is 1 and we are not storing anything into IR or DR so both these are 0.

(Refer Slide Time: 39:51)

Micro operations and control signals - Mem group

Micro operation	MW	MR	IorD	IW	DW
IR = Mem[PC]	0	1	0	1	0
DR = Mem[Res]	0	1	1	0	1
Mem[Res] = B	1	0	1	0	0

You also might here it might occur to you that there is some kind of redundancy in these signals. It may appear that you may be you to derive one of these from others or one from one of these from more than one of others so there are yes many possibilities. You might even notice that some signals can be totally omitted. For example, IW and MR seem to be identical. So we can make such observation and simplify the controller design. So it is indeed possible but we will just limit at this and not get into those details.

Finally the default here is to keep a write read both signals low, also the register load signals low and then this does not really matter. Some convenient name for these; this is fetch, this is memory read, memory write and no operation. So now, in later discussion this….. once we say fetch it would mean that this is the operation we are performing and this is the set of values we have (Refer Slide Time: 41:17) given to control signals of this particular group.

(Refer Slide Time: 41:21)

Micro operations and control signals - Mem group

Micro operation		MW	MR	IorD	IW	DW
IR = Mem[PC]	fetch	1	0	1		0
DR = Mem[Res]	m_rd	1	1	0		1
Mem[Res] = B	m_wr	0	1	0		0
default	nop	0	X	0		0

(Refer Slide Time: 00:41:29)

Micro operations and control signals - RF group

Micro operation	RW Rdst M2R AW BW

The third group is register file group and here we are talking of writing a signal RW register file write, Rdst this decides where the address comes from when you are writing; whether it is RT or RD, this tells (Refer Slide Time: 41:48) where the data to be written is coming from whether it is coming from memory or from ALU and whether we are writing into A and B register. So reading RS into A we make RW 0, Rdst don't care and 2R don't care, these Rdst and M2R will be relevant only when RW is 1 so when you are reading these do not matter we give a write signal to it and B does not require….. similarly, reading into B is similar except that we write into we make BW as 1.

Now here it is writing into register file so we will make the write signal 1, we need to define Rdst so it will be 1 in this case, M2R is 0 and we are not modifying A and B. This is writing DR into register file so again RW is 1 but this has a different value, Rdst is different because the address is coming from different points and also the values being written are different. AW BW both are 0 and default is to keep RW 0 and to keep so we do not make any change in the state here so all write signals are kept 0.

(Refer Slide Time: 43:27)

Once again you would notice that these two signals Rdst and M2R (Refer Slide Time: 43:27) are complementary of each other so one could reduce the signals. Names for these are: rs2A, rt2B, res2rd, mem2rt and no operation. So these five names I am going to use later on.

(Refer Slide Time: 43:49)

(Refer Slide Time: 43:58)

Finally, the ALU group. The signals are opc; it is a 3 bit signal sorry this is a 2 bit value which we derived from the opcode. What goes to ALU finally is a 3-bit signal which will look at opc and the function bits and that part of the circuitry will be totally unchanged. Then we have the multiplexor control signals A source 1 and A source 2 and signal which controls writing into result register.

So a micro operation which we saw earlier is appearing here also because doing PC gets PC plus 4 influences the PC group of signals as well as the ALU group of signals because

we need to ensure that addition is done here. We will look at opc later; look at the source and get back to this diagram.

(Refer Slide Time: 00:45:06)

Control signals in multi-cycle DP
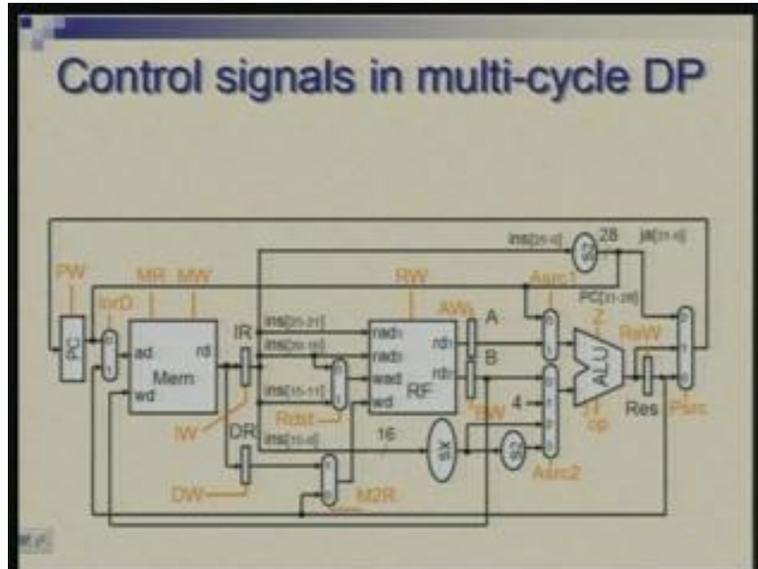
A source 1 has a choice of PC and A, A source 2 has a choice of B 4, this offset for load store and offset for branch. So these are the four possibilities here and two possibilities there. So let me put all these together actually.

(Refer Slide Time: 46:02)

Micro operations and control signals - ALU group

Micro operation	opc	Asrc1	Asrc2	ReW
PC = PC + 4	0	0	1	0
Res = A op B	2	1	0	1
Res = A + sx(IR[15-0])	0	1	2	1
Res = PC + s2(sx(IR[15-11]))	0	0	3	1
if (A == B) PC = Res	1	1	0	0

So, A source 1 has value 0 or 1; 0 for pc here and here and 1 for A, A source 2 has four possibilities, 0 for B that is here 1 for 4, 2 for load store offset and 3 for branch offset.

For this operation again we are comparing A and B so it is like this as far as source are concerned 1 and 0 this and this. This is indicating (Refer Slide Time: 46:48) whether we are writing into Res or not. So in these three steps we are writing. Here we are not writing into Res, here we are not writing into Res and this opc encoding is same as what we had done earlier. For those instructions where we have to simply perform addition without looking at anything else we make it 0 so we had done this for load store instruction and now even for these address calculations we will use 0 because our logic would be that whenever there is a 0 here that ALU controller would ensure that ALU performs the addition and one would mean that it performs subtraction unconditionally and two it would mean that we look at the function bits. So same encoding is used and accordingly we fill this up.

(Refer Slide Time: 47:48)

Micro operations and control signals - ALU group

Micro operation	opc	Asrc1	Asrc2	ReW
PC = PC + 4	0	0	1	0
Res = A op B	2	1	0	1
Res = A + sx(IR[15-0])	0	1	2	1
Res = PC + s2(sx(IR[15-11]))	0	0	3	1
if (A == B) PC = Res	1	1	0	0
default	X	X	X	0

The last case is the default where these are don't care does not matter what ALU does as long as the result is not written anywhere so we ensure that ReW is 0 and ALU may do something which is don't care. So now with this done we have seen the relationship between micro operations which we picked out of that flow chart and how they assert various control signals; what values they imply for various control signals. The names of these are PC increment, arithmetic, memory address, PC address, branch and no operation.

(Refer Slide Time: 48:35)

Micro operations and control signals - ALU group

| Micro operation		opc	Asrc1	Asrc2	ReW
PC = PC + 4	PCinc	0	1		0
Res = A op B	arith	1	0		1
Res = A + sx(IR[15-0])	Maddr	1	2		1
Res = PC + s2(sx(IR[15-11])	Paddr	0	3		1
if (A == B) PC = Res	branch	1	0		0
default		X	X	X	0

(Refer Slide Time: 00:48:53)

Control states and micro operations
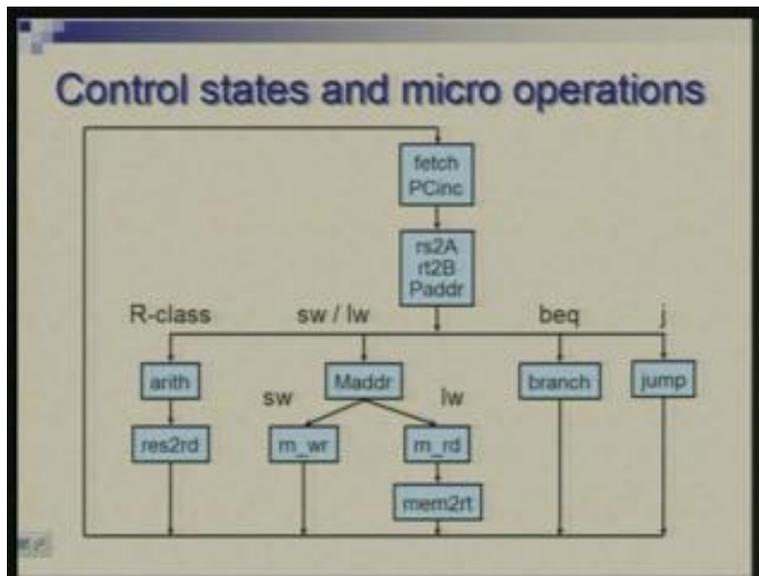
So now I can tabulate these. Before that let me redraw the diagram with these new symbols put in the boxes. So instead of those assignment statements I have replaced those with the new micro operation symbol which I have described in previous few slides. So for example, in the first cycle I am doing fetch and PC increment, in the second cycle I am doing rs2A and rt2B and Paddr; all these three are done concurrently within the second cycle and so on. So all these signals have been all these micro operations have been put into appropriate states. Also, these states could be numbered: cs0 to cs9.

(Refer Slide Time: 49:44)
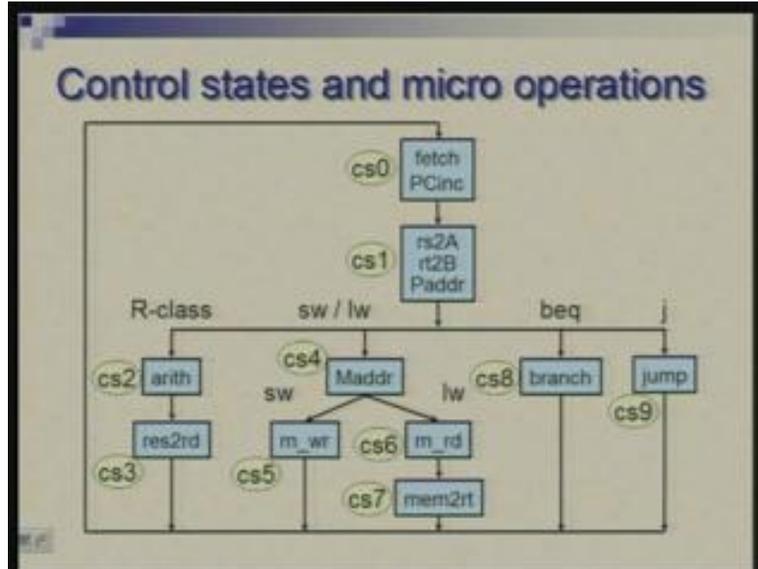
Now I need basically two tables: one table will define for a given control state what micro operations I perform which will also directly imply what are the signal values for various control signals that is one table. Next table would define that given a control state what is the next control state and this transition may be conditional; it will depend upon the way I am bifurcating so it will depend upon the opcode. So let us see both these tables one by one. So first, we will see relationship between control states and signal values.

(Refer Slide Time: 00:50:24)

So, I am not listing signal values here; I am only listing the micro operation in the particular group. So these are the four groups I identified: PC group, Memory group, RF

group and ALU group and in cs0 the operation I need to perform is PC increment this actually shows up here also as I mentioned it requires to control signals at both the ends and in memory group the operation is fetch.

In cs1 I am fetching these operands and calculating branch address. In cs2 this is the first distinct state for R class instruction so I perform the arithmetic operation here so the R class instructions go through cs0 cs1 cs2 and cs3. In cs3 the value gets written. And all those which I want to keep inactive you will find there is no offsetting there. So here the result is being written into register file (Refer Slide Time: 51:41).

(Refer Slide Time: 51:47)

Control states and signal values

	PC grp	Mem grp	RF grp	ALU grp
cs0	PCinc	fetch	nop	PCinc
cs1	nop	nop	m2A,rt2B	Paddr
cs2	nop	nop	nop	arith
cs3	nop	nop	res2rd	nop
cs4	nop	nop	nop	Maddr
cs5				

Then c4 is the common state for load store here memory address gets calculated, then c5 completes the store operation we have a memory right here, c6 performs memory read operation and c7 performs transfer of data from memory to register file, c8 completes the branch instruction so again a branch operation branch micro operation shows up in PC group as well as in ALU group because it requires a comparison here and it needs to change the state of PC so both these get influenced. And c9 completes the jump instruction so it is influencing this pc group.

(Refer Slide Time: 52:33)

Control states and signal values

	PC grp	Mem grp	RF grp	ALU grp
cs0	PCinc	fetch	nop	PCinc
cs1	nop	nop	rs2A,rt2B	Paddr
cs2	nop	nop	nop	arith
cs3	nop	nop	res2rt	nop
cs4	nop	nop	nop	Maddr
cs5	nop	m_wr	nop	nop
cs6	nop	m_rd	nop	nop
cs7	nop	nop	mem2rt	nop
cs8	branch	nop	nop	branch
cs9	jump	nop	nop	nop

What I can do but I will not do that here is that each of these micro operation symbol I replace it by a bit vector that bit vector defines the relevant control signals. Once I have that I also encode these nine states in binary so you can use 4 bits to encode this state. I form a truth table where this is the input, the code of the state is the input and the bit vector which I write here are the outputs. So these are control signals which go from controller to the data path.

The second part of the control design is how control state transitions take place. So, again for each of the states I am defining the next state but the next state could be different depending upon what is the opcode value. So from cs0 I am unconditionally going to cs1 in all cases. From cs1 one goes to cs2 or cs4 for load store it is common again, for branch it is cs8, for jump it is cs9.

(Refer Slide Time: 53:58)
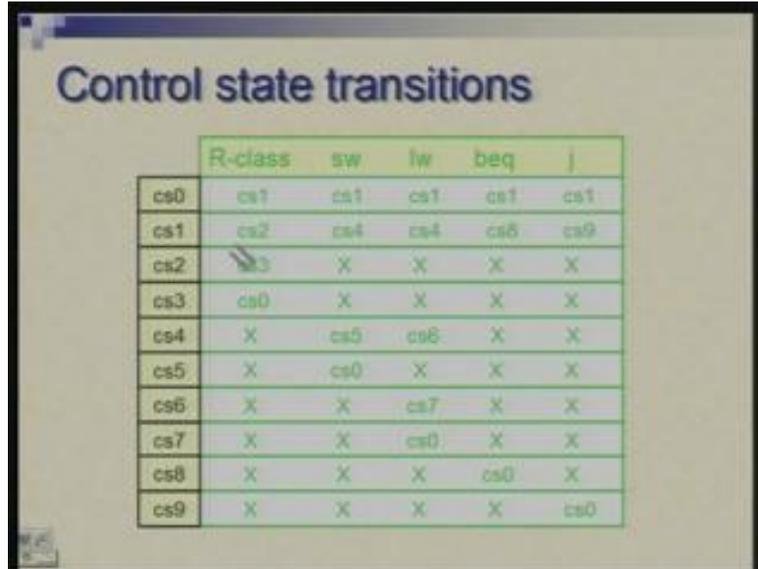
Control state transitions

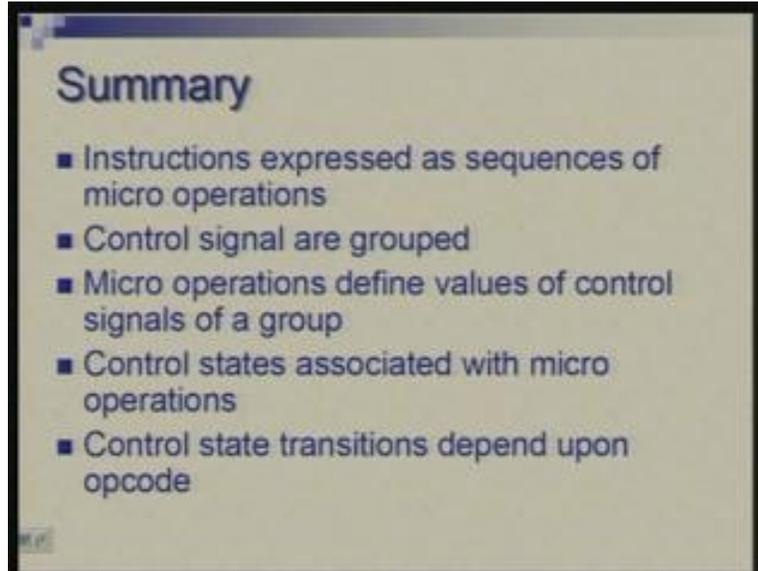|	R-class	sw	lw	beq	j
cs0	cs1	cs1	cs1	cs1	cs1
cs1	cs2	cs4	cs4	cs8	cs9
cs2	cs3	X	X	X	X
cs3	cs0	X	X	X	X
cs4	X	cs5	cs6	X	X
cs5	X	cs0	X	X	X
cs6	X	X	cs7	X	X
cs7	X	X	cs0	X	X
cs8	X	X	X	cs0	X
cs9	X	X	X	X	cs0

From cs2 which is for R class instruction go to cs3 and these conditions will not occur. Once you have gone to cs2 you know that it is R class instruction and others are not relevant. So you can see that R class instruction goes through cs0 1 2 and 3 and back to cs0.

For store instruction we start with cs0 go through cs1 cs4 cs5 and then cs0. For load it is cs0 cs1 cs4 then cs6 cs7 and then cs0. For branch it is 0 1 and 8 and then 0 for jump it is 0 1 9 and then 0.

So now what kinds of truth tables I have here?
Here the opcode and 4 bit of the control state these form the input of one combinational block and the next state value or the 4 bits which encode the next state are the output. So if you implement these two tables the tables shown here and the tables shown in the previous slide namely this one you can use two PLAs to implement this so these two PLAs plus one register holding the control state 4-bit register hold in the control state will form the controller. I will draw that picture next time and look at some alternatives. So today we will stop at this.

(Refer Slide Time: 00:55:42)

Summary

- Instructions expressed as sequences of micro operations
- Control signal are grouped
- Micro operations define values of control signals of a group
- Control states associated with micro operations
- Control state transitions depend upon opcode

Let me just summarize that we saw how instructions get divided into sequences of micro operations; how we group the control operations control signals and then define the relationship between micro operation and control signals then we associated control states with the micro operations and we also identified control state transitions, thank you.