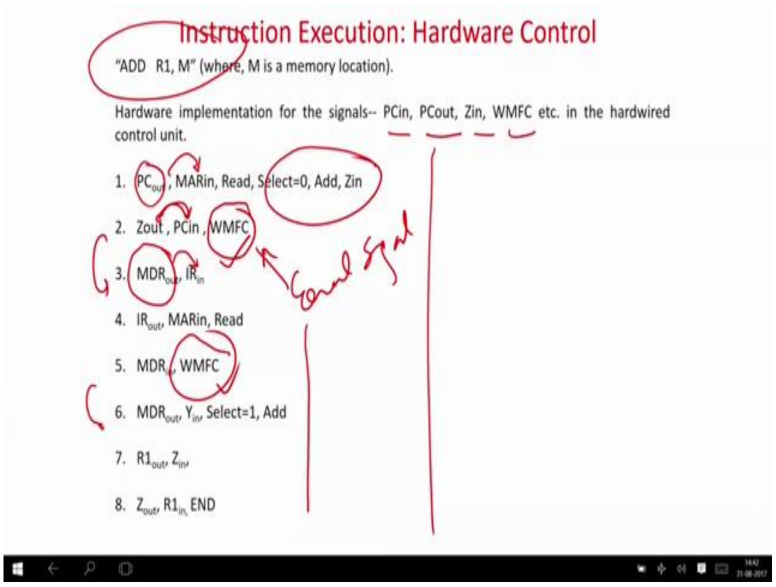
(Refer Slide Time: 19:48)



Instruction Execution: Hardware Control

"ADD R1, M" (where, M is a memory location).

Hardware implementation for the signals-- PCin, PCout, Zin, WMFC etc. in the hardwired control unit.

1. PC_{out}, MARin, Read, Select=0, Add, Zin
2. Zout , PCin , WMFC
3. MDR_{out}, IR_{in}
4. IR_{out}, MARin, Read
5. MDR_{in}, WMFC
6. MDR_{out}, Y_{in}, Select=1, Add
7. $R1_{out}$, Z_{in},
8. Z_{out}, $R1_{in}$, END

So, first let us take an example called $ADD\ R1, M$ that is you are taking the memory location value $R1$ dumping it to $R1$ after adding it to whatever the content in $R1$ if you look forget about last class we have discussed that these are the series of control instructions, what are the signals involved? Program counter in, program counter out, Z_{in}, MFC all the signals whichever we are listed here are actually utilized for this micro-instruction, sorry; the micro-instructions which will be involved in this macro instructions. So, they are all will be generated.

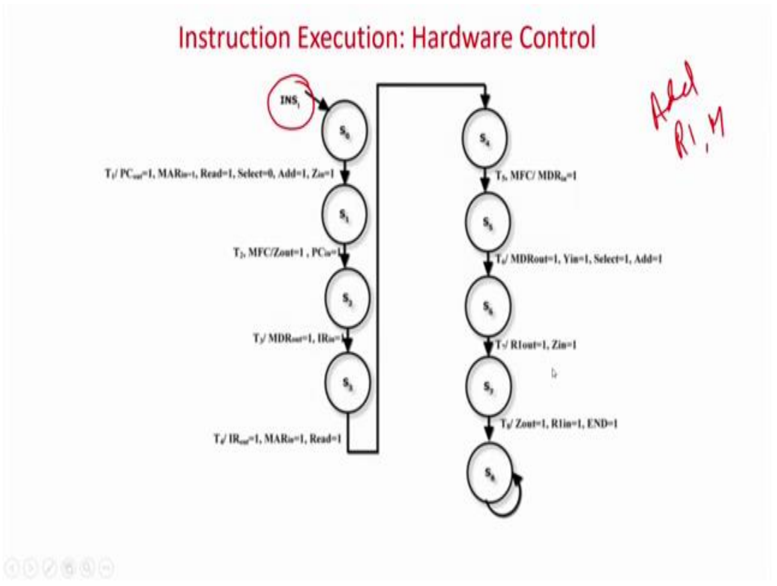
So, now, first we will say if we remember that first was program counter out that is the PC value will be fed to the memory address register and then you will make it select zero, Add, Z_{in}. This part corresponds to incrementing of the value of PC. So, whenever I say $PC = Z_{out}$; that means, the PC is incremented, sorry this way. So, whenever Z_{out} means the value of incremented value of PC will be dumped into PC output, because Z output actually was storing the value of accumulated sorry the value of the ALU which was added the value of program counter plus the constant.

Now, we have to put it in PC that now PC is updated. Now, we have to it's important we are waiting for an external signal this is an external signal. So, this is an external signal. So, we have to wait for the external signal only once when it is ready in stage three basically what you can do? The instruction which is now in MDR can be loaded to register in and similarly you can go ahead. So, again there is a external signal and all others are internal signals like PC_{out},

MAR_{in}, read they are all signals which are generated and here we are waiting for something which is again a condition which I am depending on which comes from an external one.

So, all these like $PC_{out}, MAR_{in}, read, select\ 0, ADD, Z_{in}$ they are the control signals which you have to generate and then $WFMC$ is something on which you have to wait till you can go to the third stage. Similarly from the 5th stage to go to the 6th stage basically you have to again wait for the $WFMC$; that means, this round this special this two are basically inputs which on which the movement of the final state machine will depend and all other signals are basically being generated which has to be generated by the your hardware control unit which in this case is a finite state machine. Now, we will first basically see the circuit and then we will go for the discussion.

(Refer Slide Time: 22:03)



Instruction Execution: Hardware Control

INS1
S0
T1/ PCout=1, MARin=1, Read=1, Select=0, Add=1, Zin=1
S1
T2, MFC/Zout=1 , PCin=1
S2
T3/ MDRout=1, IRin=1
S3
T4/ IRout=1, MARin=1, Read=1
S4
T5, MFC/ MDRin=1
S5
T6/ MDRout=1, Yin=1, Select=1, Add=1
S6
T7/ R1out=1, Zin=1
S7
T8/ Zout=1, R1in=1, END=1
S8
Add R1, M

So, let us assume that this $Add\ R1, M$ whatever may be the opcode for this corresponds to first instruction. So, what is going to happen INS_1 will be 1; that in the decoder the first line will be 1 which corresponds to this instruction which is the first instructions, so whenever the INS is 1, because it's a decoder. So, only one line will be 1. So, only this finite state machine will be invoked, if you look at the initial stage it's input is INS_1.

So, only if the value of $INS_1 = 1$, then this machine will invoke and you know there is already a decoder. So, based on the input of the opcode only one of this machines will be invoked like there will be another machine for INS_2. Another machine for INS_3 ... up to INS_m, so based on which instruction is in the instruction decoder the opcode it will be correspondingly invoking only one finite state machine, because correspondingly this input is equal to 1.

Now, next what let me zoom this part so it will be clear. So, next is what I am doing? So, you are invoking this finite state machine based on the in which based on the output of the instruction decoder $INS_1 = 1$. Next what? Next I am going to go to state 1. So, what is the output? Initial state is there from state 0 to state 1 what happens basically, you should know that now I should be in state T_1 that is from 0 next state has come that is counter has now become one. Then what are the outs if you remember there is only one thing over here after initial state basically you need not depend on any other input only thing is that state 1 corresponds to the fact that at present the instruction is INS_1 that is $ADD\ R1, M$ register to memory. Then next is nothing just you need to wait till T_1 comes, T_1 comes means that is sequence counter has now become 1.

So, what is that now after that there is no other input required only time has passed the clock has come that from stage 0 to stage 1 you can go and what are the signals out $PC_{out}, MAR_{in}, read\ 1, select\ 0, ADD\ 1, Z_{in}\ 1$. So, all these signals will be correspondingly generated at this state and all other signals which are not shown like MAR_{in} is 1. So, what about MAR_{out} it will be equal to 0 like $PC_{out} = 1$. So, what about PC_in it will be 0. So, whatever I have not shown here are made 0 now it is important. So, now, in this case you have said that I want to read the memory in the memory address register you have given the value of PC; that means, you are going to fetch the instruction.

Second state is interesting second state is interesting basically. So, when I am in state one again you are waiting till you go to the second state that is second clock pulse then the counter has become two. Not only here you have to again wait here for another external symbol that is $WFMC$ that MFC you have to wait this is another signal which will be coming from the memory to the bus it was going to tell that now the memory has been read everything is fine now you can go ahead. Only after this you can go to the next state. So, what is the next state; you are saying that $PC_{in} = 1$ and $Z_{out} = 1$ that is you are going to dump the value of program counter with an updated value from Z which is nothing but the constant value of ALU, but before initially there is only one input which we will be depending on that is your clock count that is state one, here you are depending on two that is clock counter as well as you are waiting till the memory is ready.

So, whenever the memory is ready you can go to the next state S2, there is nothing else you to depend on just the timer; because already the memory signal has been ready in the previous state just you dump the value of memory data register to the instruction register. After that just the sequence because there is no if then else condition you did not wait for any memory read here. So, the memory data register out; that is basically instruction has been now loaded to the instruction register.

For state three nothing you have to do just you have to wait for the next clock pulse which will make the state equal to 4. Here instruction register out will go to the memory address register in, because you have to fetch another operand from the memory if you remember your instruction is $Add\ R1, M$ the next instruction is available in the memory location. So, what you are going to do instruction register out that is m in the instruction register is dumped into the memory address register and you are making the register mode as 1.

So, these are the control signals which will be generated all other not mentioned like memory out, R_{in} they are all zeros then again I go sequentially to the next state, here also I do not need to depend on anything else except it's important over here I have given a read signal to the memory. So, I cannot only depend on the input. So, only input here is T_5; that is state 5 is going to come the counter is 5, I cannot depend only on that here again I have to wait till the memory gives a green signal that I have done you can read it.

So, whenever it says MFC; that means, I am done. So, then you can go for here actually MDR_{in} is 1; that means, you are going to generate the signal that I want to read the memory; because till here it is memory address you have given the value of the register value M and you are saying $read = 1$. So, you have set the the memory in a read mode. So, this one says that memory is being read, a green signal has been there. Then only you can give the corresponding signals like here already MFC has been 1 right, MDR_{in} means you are going to get the value of the memory into memory address register, but now you cannot read here, because you cannot read at this at this stage you cannot read basically the value of MDR, because I am waiting for the memory location to be done. Here I am giving a read signal, but when I am going to state six; that means, already this has been over long back. So, I can go for $MDR_{out} = 1$ and all other similar procedure like this signal has been done.

So, if you remember for all other states, similarly I can complete all the sequences of this state machine design like for 6 this is the state 6 that counter is 6. I will generate the corresponding control instruction 7, I will generate the control signals correspondingly 8; I will generate the corresponding signals correspondingly. Finally, $END = 1$ means you have to stop this set of micro-instructions and the macro instruction is over.

So, what is to be emphasized it's very easy to understand only some clocks are there first is this machine is invoked that is the initial state condition enabling is INS_1 that is based on the opcode only when the signal is one you can go to the corresponding state machine. If there are m instructions m such state machines will be there and based only on the output of the instruction register via the decoder you can invoke that corresponding state machine.

Now, for all cases generally if you do not wait want to wait for any kind of an external input like; your condition flags or your memory values output of the memory that is memory is ready or there is some signals like you want to wait that if the flag is 0 or flag is 1. So, if no such conditions are there you just need to wait for the time stage 1, stage 2, stage 3 these are generated by the clock and the FSM counter, go to stage one and generate all the corresponding signals which are required to do that only here like whenever you have given a read signal to the memory.

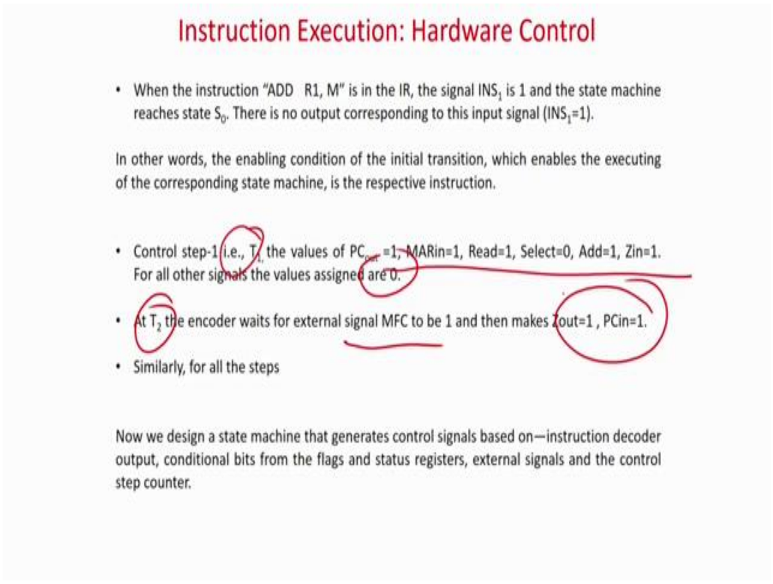
So, you have to wait till the memory says that I am ready. So, in that case the input from this state to this state the transition will depend on two inputs one is the state and one is the external signal and then you generate the corresponding signals like on this case it will be similar, but here again I have to wait for the memory to be ready, because here I have given a memory read this is a memory read state that T_4 and you are going for IR_{out} that is the instruction register you want to read from the operand from the memory location M.

So, in the memory register address register you are giving the value M and you want to read it. So, at least you have to wait for some amount of time till you can get the value of memory data out to your some kind of the another operand y where it will be stored. So, this ready can happen only when if the MFC is ready. So, only in that way it can be done ok. So, in this way you can explain the whole thing. So, if there are 6, 7, 8 are just sequential part and then finally, you are going to generate the END out of it.

So, this is the finite state machine corresponding to the corresponding to the set of macro micro-instructions corresponding to the macro instruction $R1, M$. That is you first write down these control steps control signals steps then you find out what are the inputs. There are two inputs only here $WMFC$; $WMFC$ and all others are signals to be generated and of course, 1, 2, 3, 4, 5, 6, 7, 8 are the sequence numbers. So, you have inputs like sequence number as well as $WFMC$.

So, these are the two inputs and for all other cases you have to generate. So, similarly you can map this to the finite state machine, I have shown you and then you have to just go for finite state machine base synthesis and your job is done how to generate a finite state machine from a may be the finite state machine design how you can generate in terms of gates and flip flops is just a simple digital design fundamental which you can read and go back and read your second semester textbooks this is what is written in written in the language you can read through the slide.

(Refer Slide Time: 30:54)



Instruction Execution: Hardware Control

- When the instruction "ADD R1, M" is in the IR, the signal INS_1 is 1 and the state machine reaches state S_0. There is no output corresponding to this input signal (INS_1=1).

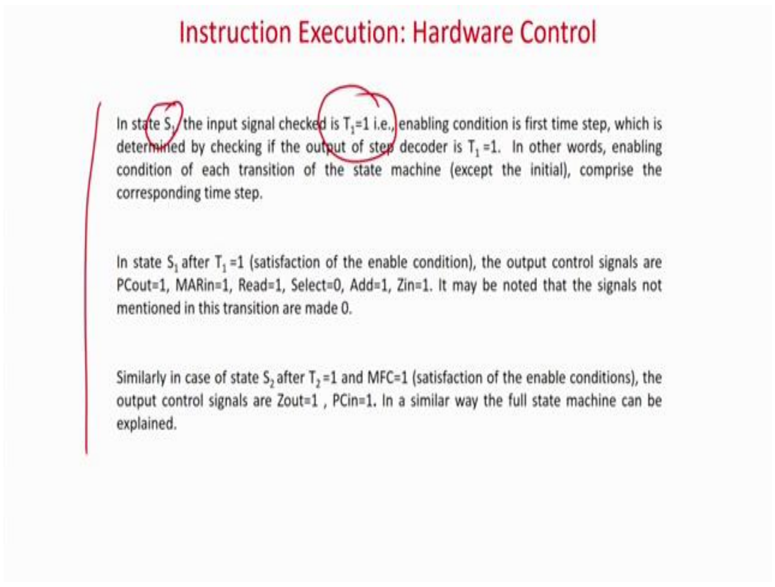
In other words, the enabling condition of the initial transition, which enables the executing of the corresponding state machine, is the respective instruction.

- Control step-1 i.e., T_1, the values of PC_{out} =1, MARin=1, Read=1, Select=0, Add=1, Zin=1. For all other signals the values assigned are 0.
- At T_2 the encoder waits for external signal MFC to be 1 and then makes Zout=1 , PCin=1.
- Similarly, for all the steps

Now we design a state machine that generates control signals based on—instruction decoder output, conditional bits from the flags and status registers, external signals and the control step counter.

Basically and you will get a very clear idea like say in control step one the values are this 1. So, you have to generate these values directly and all other signal values are 0, which I have already told you at T_2 you have to wait for the external signal MFC to be 1. So, basically there are two inputs for the second stage; that is your clock counter stage counter as well as the signal from the memory and then you generate the $PC = 1$. Similarly in this way you can generate the whole circuit whole finite state machine and then you can synthesize this right.

(Refer Slide Time: 31:21)



Instruction Execution: Hardware Control

In state S_1 the input signal checked is T_1=1 i.e., enabling condition is first time step, which is determined by checking if the output of step decoder is T_1 =1. In other words, enabling condition of each transition of the state machine (except the initial), comprise the corresponding time step.

In state S_1 after T_1 =1 (satisfaction of the enable condition), the output control signals are PCout=1, MARin=1, Read=1, Select=0, Add=1, Zin=1. It may be noted that the signals not mentioned in this transition are made 0.

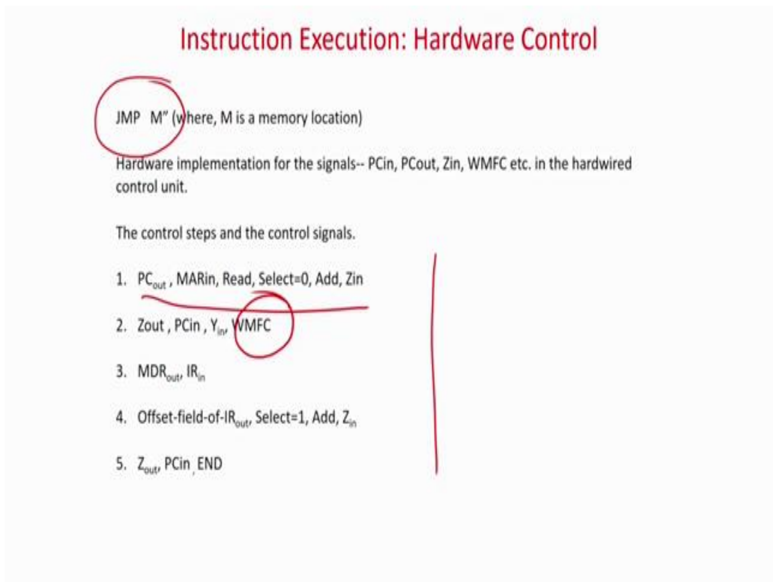
Similarly in case of state S_2 after T_2 =1 and MFC=1 (satisfaction of the enable conditions), the output control signals are Zout=1 , PCin=1. In a similar way the full state machine can be explained.

This is this is about basically your state generation in state one, basically you have to you have to check whether the T_1 that is state 1 is equal to 1. So, that is this slide is basically saying that from one state to another state the transition depends always on something called the state counter that is T_1, T_2 T_3; that is the states that it will depends on state counter that I can go from state 1 to state 2; only if the state variables have been incremented this is just a finite state machine based implementation.

So, whenever you want to generate implement a finite state machine you remember that always the there are you have to increment the states like 000, 001, 101 if you generate the forward counter. So, that that only you can translate into a sequential machine in states will go through this is just a simple digital design fundamental. So, basically this state slide is showing that for all the cases one primary input for this hardwired control is the state counter T_1, T_2, T_3 they are nothing, but the state variable values 0, 1, 2, 3, 4, 5, 6 along with that sometimes you will have extra inputs which in this case the output of the memory sometimes it can be flag register values also ok.

So, now basically you are going to as I have told you we are also see basically how a flag register can be an input. Till now the last instruction was a simple non control based implementation just a flow, but here we are going to see two type of instruction in which case its an unconditional branch and one will say the conditional jump. So, conditional jump actually the corresponding instruction we will have another input which will come from the flag register so that you can get a basic idea.

(Refer Slide Time: 32:56)



Instruction Execution: Hardware Control

JMP M" (where, M is a memory location)

Hardware implementation for the signals-- PCin, PCout, Zin, WMFC etc. in the hardwired control unit.

The control steps and the control signals.

1. PC_{out} , MARin, Read, Select=0, Add, Zin
2. Zout , PCin , Y_{in}, WMFC
3. MDR_{out}, IR_{in}
4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}
5. Z_{out}, PCin , END

So, jump to M as you have already seen these are the control signals involved there is nothing much to discuss, you just look at the other last previous unit then you can get through what are the signals to be generated. So, these are all the signals to be given as output this is 1 input we have to wait for from the memory and that's it and all others are signals which has to be generated right or as the output signal. So, what will be the step? So, again importantly this is assume that jump M is signal number instruction number 2.

So, jump unconditional to M in signal number 2; so, only when the opcode corresponding to jump M comes; then only this INS_2 will be equal to 1. So, only the initial state input value will be satisfied only for this finite state machine for all instruction there are different different finite state machines, but whenever you are going to give jump M in the instruction register the corresponding opcode will actually satisfy only the input condition for this finite state machine. So, this will be input and it will not satisfy the initial condition state for the previous instruction or any other instruction in that matter.

So, only this instruction will be activated. So, again as I told you stage 1 so, counter is one you generate all the signals and go to state 1. State 2 basically similar two is the input and you have to wait for the memory to be ready, because you are going to read the memory value as the instruction is in the first stage. So, whenever it is ready then only you can actually start the memory read. So, it is generating the signals and the third stage, because in this stage you have already verified that the memory is ready. So, you are saying that $MDR = R_{in}$. So, this state says that memory is ready. So, you can actually dump the value of the memory instruction.

T_4 what basically as you see the jump unconditional. So, you have to take the offset value and actually you have to put it in Z_{in} already we have seen. So, in this case you have to take the value of offset and $select$ 1 add 1, Z_{in}; that means, what you are going to do you are going to get the updated value of PC you are going to add with the offset value of IR and then you will actually we are going to get the new value of the program counter that is the jump address and you will dump that value to PC and your job is done.

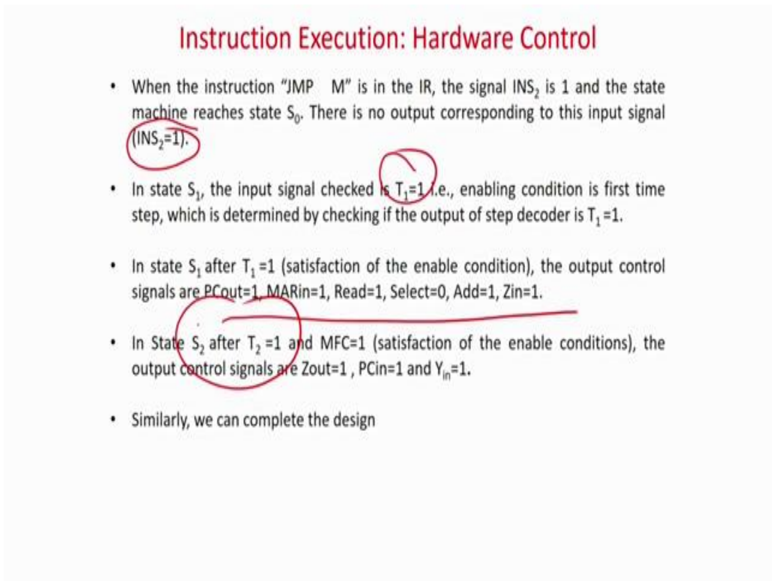
So, basically, but here there is no condition, so whatever maybe state there is no input required from any flag register just you have to take the value of offset value of IR you have to add it with the program counter value which is already saved in if you look at it. So, the PC_{out} and Z_{in} and of course which is saved in some temporary register in that manner you have to just get the value added and it will be the new value of the PC so. In fact, there is nothing to depend upon. So, we just say that IR_{out} that is instruction.

Of course, the offset is taken as the output $select\ one, add\ one$, Z_{in} if you remember it means that I am selecting the Y variable not the constant variable to be added to the there is an updated value of PC which will be in the data bus which will be present in the bus and the output will be saved in Z_{in} that is actually equal to PC plus offset value of PC which is nothing, but equal to jump the variable M will come to the PC the value of M will come to the PC that is state 4.

So, once it is the condition we will come over here then actually a Z_{out}, PC_{in}; that means, there is a Z_{out} here is nothing but the M which will be dumped to the program counter and it’s end, that means in this case we are just updating the value of PC to M it was jump M. So, already we have seen that there will be just a sequence of step, because of the unconditional instruction. So, only everywhere we will depend on just the state that is whenever the counter will be four you can go over here and so, forth.

And it will be basically end the machine it is a very simple explanation which we have done to, but to appreciate the fact that when there is another input on which your state machine transition will depend will be clear when you are taking a conditional jump which we are now going to do, again you can just go through the slide which I have just told you that what will happen after which state.

(Refer Slide Time: 37:00)

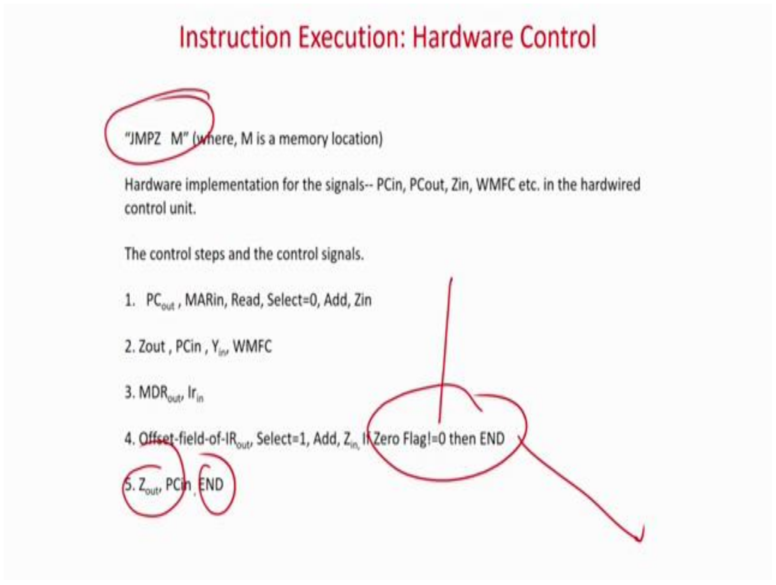


Instruction Execution: Hardware Control

- When the instruction "JMP M" is in the IR, the signal INS_2 is 1 and the state machine reaches state S_0. There is no output corresponding to this input signal (INS_2=1).
- In state S_1, the input signal checked is T_1=1 i.e., enabling condition is first time step, which is determined by checking if the output of step decoder is T_1 =1.
- In state S_1 after T_1 =1 (satisfaction of the enable condition), the output control signals are PCout=1, MARin=1, Read=1, Select=0, Add=1, Zin=1.
- In State S_2 after T_2 =1 and MFC=1 (satisfaction of the enable conditions), the output control signals are Zout=1 , PCin=1 and Y_{in}=1.
- Similarly, we can complete the design

Like for example in jump state this is the initial condition which is enabled, at T_1 state one then you are going to generate all these signals, at T_2 after the second state counter is one you are going to generate the whole signal and then the whole design can be very easily completed.

(Refer Slide Time: 37:18)



Instruction Execution: Hardware Control

"JMPZ M" (where, M is a memory location)

Hardware implementation for the signals-- PCin, PCout, Zin, WMFC etc. in the hardwired control unit.

The control steps and the control signals.

1. PC_{out} , MARin, Read, Select=0, Add, Zin
2. Zout , PCin , Y_{in}, WMFC
3. MDR_{out}, Ir_{in}
4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}, If Zero Flag!=0 then END
5. Z_{out}, PCin , END

Most interestingly now whenever I tell you that the input to the finite state machine will depend on state as well as some external inputs like your memory or your register flags then we have to take a conditional jump. So, this is a conditional jump, jump on Z to M. So, therefore if you see all other steps will be similar. Here we are saying that the offset of IR select and everything, but if flag value is not equal to 0 then end, if flag equal to 0 then you update the value of 0 PC with M, but if the flag variable is not set, then you go over here and you don't have to do basically anything or if the flag value is 0, then you have to update this that Z_{out} PC_{in} means the update value or PC that is M will be dumped to PC.

So, the program counter will have the value M and we will start executing from instruction which is present in memory location M, but if the flag value is not 0. Now from there itself we will come out and the PC will go as forward without having dumped the value of M into the PC. So, PC will be PC plus increment and it will keep on doing it. So, here this will be very interesting to see how we will do it. So, again INS_3. So, you are assuming that jump on 0 to M is instruction 3.

So, only this line will be one. So, only this finite state machine will be invoked. So in fact, you can see that for each instruction there is a separate finite state machine which will be invoked. So, you can understand that if n finite state machines will be there which will be hardcoded. So, it will take some more area, but in fact it will be extremely fast; because here you are mainly depending on the inputs which is nothing but your states. If have lot of or lot of instructions you could have optimized in this way that many of for many of the cases the initial states up to first one two three states are similar then only it is deviating.

So, I can make a more complicated type of finite state machine say for instruction 1, 2 and 3, this is the finite state machine which is synthesized; because the first three are similar. Depending on the input I can then bifurcate into some different branches then again I can do it. So, of course, we will have an optimized hardware implementation because may be the first three states are similar for everybody last two states are similar for everybody. So, I could have made a merged machine and then keep on branching and joining which will give me a better optimized machine, but it will be slower because you will be now depending on the state as well as type of instructions and again you will be joining and merging. So, this hardware size will be a bit small, but it will be a basically more slower design.

So, if I have a full flexibility then it will become a program that is your micro program based controller which we will see later, but for the time being we are actually giving dedicated hardware for each instruction and it is the fastest way of implementation. So, in this case INS_3 is enabled; that means, your instruction at present is jump on 0 to this 1 to some location, T_1 will be similar stage one you are going to generate all the signals as output, T_2 control state is two you have to wait till the memory is ready. So, that you can get the updated value of program counter from Z_out, of course you are storing the value of program count in the temporary variable Y, because you have to add PC to your offset value MDR this is very similar, instruction you are dumping to instruction register from the memory data register this is instruction fetch.

Now, important now we are saying that if T_4 of course, if zero flag is set. So, now, you see the inputs are 2, 1 is the state another one is your zero flag. So, this is an input which is coming from basically the external register sorry the register input that is the flag register this is an external input that is from the memory. So, if this state depends on two inputs, this state depends on only one input that is your state count, here also it depends on two input that is state as well as zero flag. So if the zero flag is set then what you do? You take the offset you add it to your Y and basically dump the output to Z 1.

So, Z_{in} that is actually now your Z is actually having the value of M which is offset plus the program counter, all these details we have seen in the last unit. So, now, Z is actually having the updated value of. So, if 0 is set you update the value of PC with M, and then you come over here and then what you do you dump the value of Z to program count and End and End. So, basically Z_{out} basically Z is having the value of M you dump to PC that is my $PC\_in = 1$.

So, now, the PC will be having the M and you will start executing from M, but there is another choice if the flag is not 0. So, there is another instruction another branch over here same for the same state if your counter has become 4 and if the zero flag is not set then what you do you just generate the end signal and it will actually come to the final step and look over. So, the micro-instructions will be stopped.

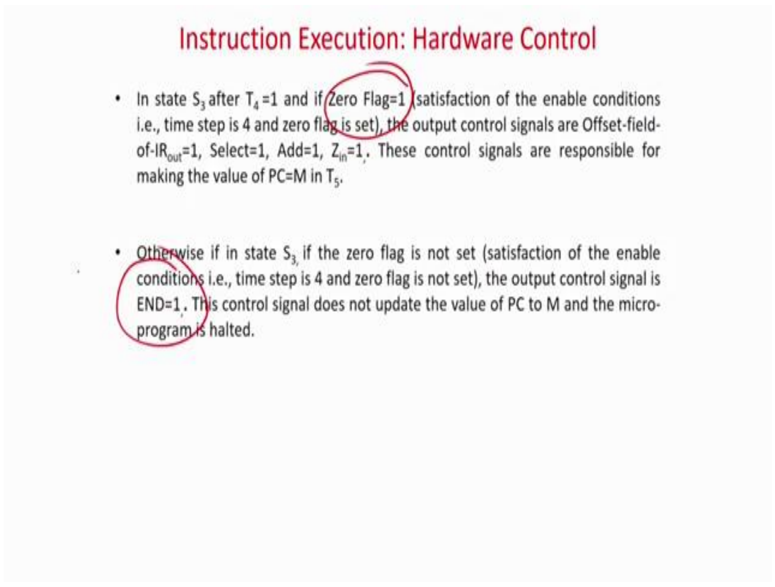
So, this shows a very nice example that if there is a conditional jump then you can have a state where there will be a bifurcation. So, here there are two types of inputs state input as well as the inputs from the flags and here then again two inputs one is the flag and one is an external input which is basically your coming from your memory that memory read has been done. So, this shows a hardwired control unit generation for a conditional jump.

So, from this discussion you can see that it is very simple to design the finite state machine which corresponds to different instructions, for each instruction basically you generate this sequence of inputs which will always be some states as well as the control signals which are to be output and for some particular cases you have to also depend on some external inputs like from the memory; if there are conditional instruction you have to look at the flag registers etcetera.

And then just you have to go for a finite state machine based synthesis which is a standard digital design fundamental and you can make a hardware out of it which will be in terms of your flip flops and gates and it will be done, but only thing is that it takes more area, because for each instruction you will have a different finite state machine if I want to merge and make an optimization of it which will be slightly having lesser area, but it will be slightly slower and we don't try to do a trade-off here.

So, what I am trying to do? whenever you have a hardware based control we dedicatedly give for each instruction a finite state machine extremely fast, but you can just see that the area overhead is higher as well as the very non flexible design, that is for a given macro instruction there are the micro instructions and the hard this finite state machines are basically hardcoded. So, again if you just look at it whatever I have explained I have given you in the thesis I will give you this description in this slide you can read through it like that is the initial state is not the enabling condition is this, in T_1 state these are the signals and so, forth and T_2 you have to wait for $WFMC$ and you have to go through and most importantly you have to actually think about this state where we depend on the output of the flag registers. So, that is what has been discussed over here.

(Refer Slide Time: 43:57)



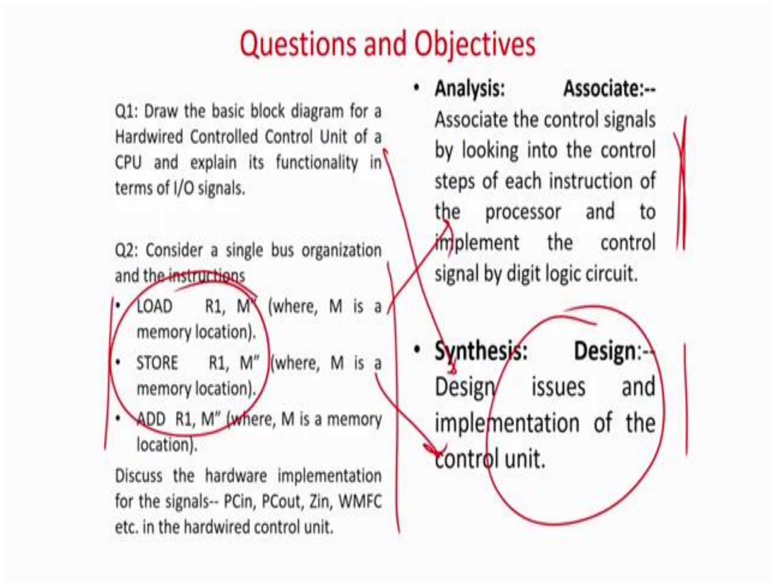
Instruction Execution: Hardware Control

- In state S_3 after T_4=1 and if Zero Flag=1 (satisfaction of the enable conditions i.e., time step is 4 and zero flag is set), the output control signals are Offset-field-of-IR_{out}=1, Select=1, Add=1, Z_{in}=1. These control signals are responsible for making the value of PC=M in T_5.
- Otherwise if in state S_3, if the zero flag is not set (satisfaction of the enable conditions i.e., time step is 4 and zero flag is not set), the output control signal is END=1. This control signal does not update the value of PC to M and the micro-program is halted.

That, if the zero flag is set which is one of the inputs what happen; and if the zero flag is not set you directly go to END.

So, with this we come to the end of this unit and just have a little look at some of the sample questions like for example.

(Refer Slide Time: 44:14)



Questions and Objectives

Q1: Draw the basic block diagram for a Hardwired Controlled Control Unit of a CPU and explain its functionality in terms of I/O signals.

Q2: Consider a single bus organization and the instructions

- LOAD R1, M (where, M is a memory location).
- STORE R1, M" (where, M is a memory location).
- ADD R1, M" (where, M is a memory location).

Discuss the hardware implementation for the signals-- PCin, PCout, Zin, WMFC etc. in the hardwired control unit.

- **Analysis: Associate:--** Associate the control signals by looking into the control steps of each instruction of the processor and to implement the control signal by digit logic circuit.
- **Synthesis: Design:--** Design issues and implementation of the control unit.

We say the first question is draw the basic block diagram for a hardwired controlled units of a CPU and explain its functionality the theoretical question if you are able to answer this question you should be able to do it; because we have both discussed using examples as well as theory of how a hardwired control unit work if you are able to solve the question you will be are basically within the objectives of associating control signals by looking at the control steps and basically the design and design issues and implementation of a control unit.

So, mainly this actually focuses on this one and also slightly on this in objective, because you have to also know what control signals are depending on what, but mainly if you were able to explain the whole control unit in terms of block diagram you are able to synthesize the idea of how to design a control unit.

Then by different examples like $LOAD\ R1, M$; $STORE\ R1, M$; $ADD\ R1, M$ different types of macro instructions are there then I ask you to design a hardwired based controller out of it. So, by if you are able to solve this question of course, you are actually meeting both the objectives because the second objective is the design objective where you are asked to design a micro controller oh, sorry; I mean say finite state machine based controller for a given set of macro instructions and of course, the you have to also have a good analysis idea, because you have to associate different control signals with different micro instruction steps.

So, if you are able to solve this your job is done. So, with this we come to the end of this unit from next unit on words basically; we will try to have a look at slightly more integrated details like what happens if you have multiple buses? How things you are going to change? How many less number of steps are there? How the finite state machine synthesis will change? So, we will be having a look at multiple bus architecture may be a 2 bus or a 3 bus architecture and also we have to look in details about your very importantly your micro program based control because you have seen that in FSM based control it is very fast, but everything is hardcoded and non-flexible.

So, we want to give more flexibility to it because you can see that many of the instructions have several parts has common. So, if we want to give some commonality and some flexibility we will go for micro program based design and then also in the end we will see how all the things get changed if you have a multiple bus architecture we just give an idea of it.

Thank you.