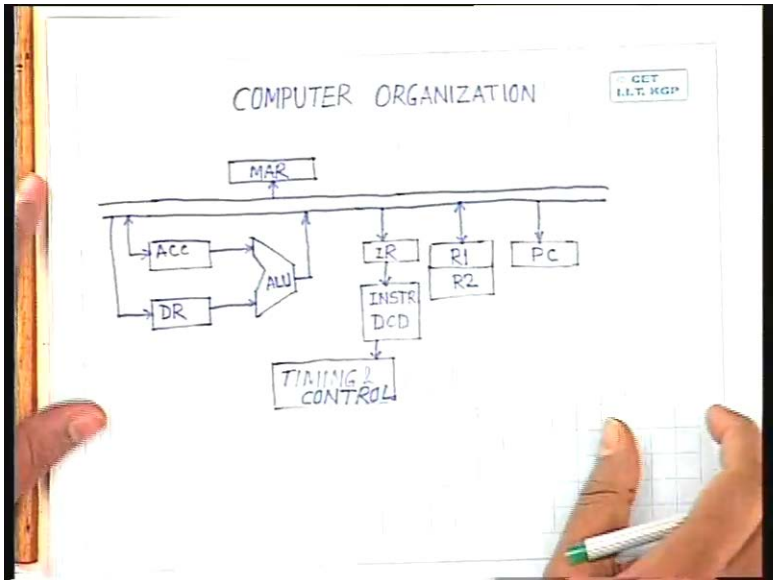
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic & Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 04
CPU Design: Tirning & Control

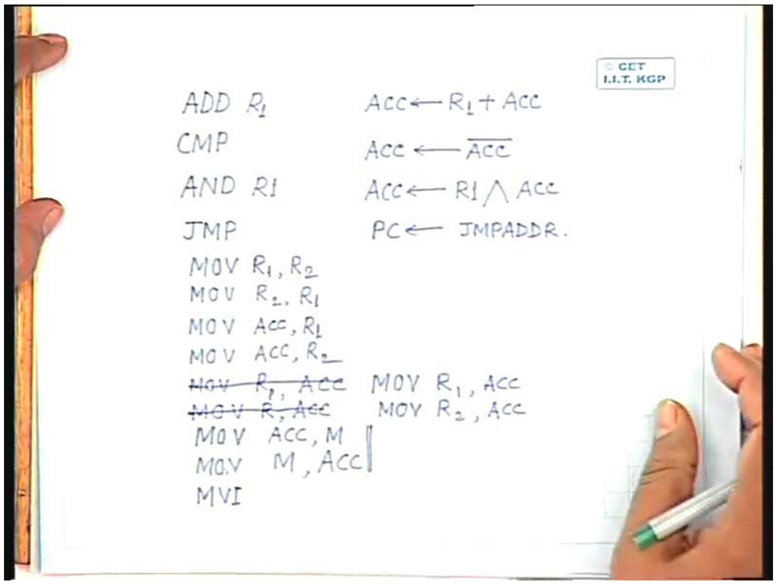
We have started discussion on designing of a simple CPU and the CPU architecture that we have considered is like this, having only two registers R_1 and R_2.

(Refer Slide Time: 00:01:16 min)



In addition to accumulator and data register, it is having an ALU, program counter, instruction register, instruction decoder, timing and control unit and a memory address register. For designing the CPU, we have considered few instructions around which the CPU will be designed and the instructions that we have said is like this.

(Refer Slide Time: 00:01:47 min)



ADD R_1 $Acc \leftarrow R_1 + Acc$

CMP $Acc \leftarrow \overline{Acc}$

AND R1 $Acc \leftarrow R1 \wedge Acc$

JMP $PC \leftarrow JMPADDR.$

MOV R_1, R_2

MOV R_2, R_1

MOV Acc, R_1

MOV Acc, R_2

~~MOV R_1, Acc~~ MOV R_1, Acc

~~MOV R_2, Acc~~ MOV R_2, Acc

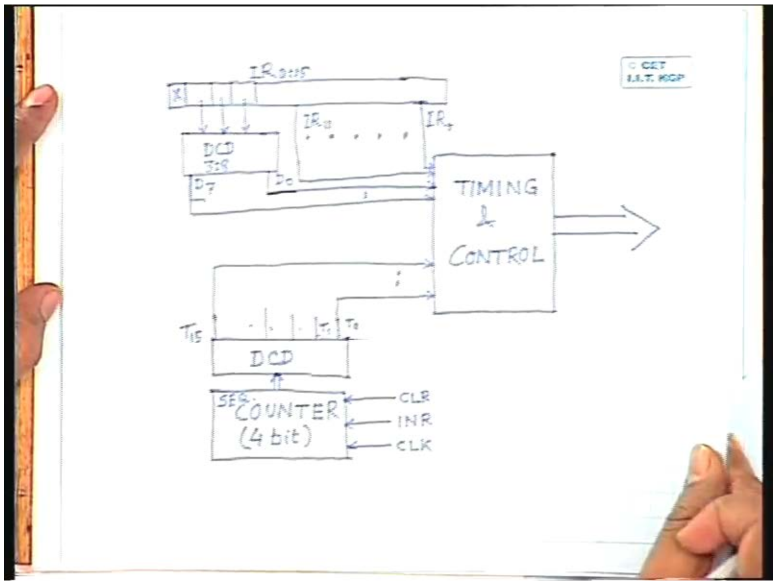
MOV Acc, M

MOV M, Acc

MVI

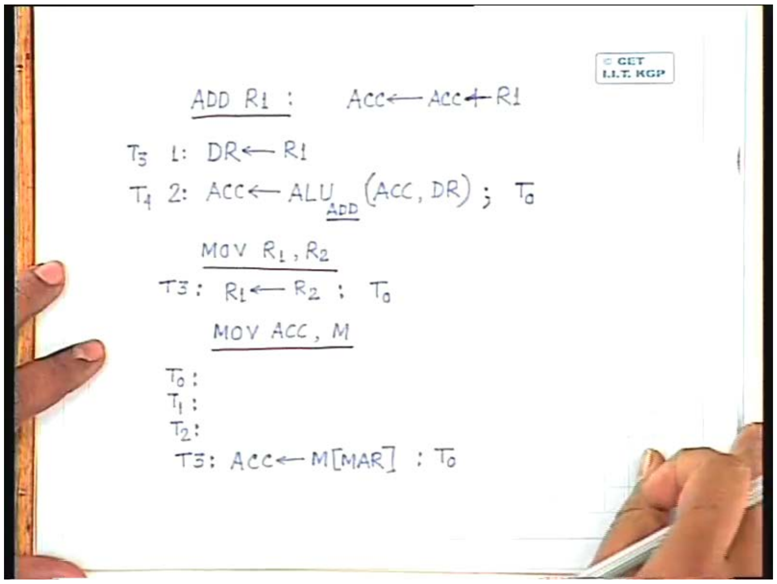
It is having an add instruction add R_1 then complement and logical instruction and R_1, one jump instruction and few MOV instructions. Then we have seen that the block diagram of the timing and control unit of this particular CPU will be something like this.

(Refer Slide Time: 00:02:07 min)



From the instruction register, the instruction opcode will go to the instruction decoder. Similarly we have a sequence counter to generate different time states of the CPU. The counter output is again going to a decoder. All the decoder outputs that is sequence counter decoder output and the instruction decoder output they are going to the timing and control unit and the timing and control circuit will generate the timing clock in the sequence that is required. So for that instead of taking the entire instruction set that we are considering, we have tried to see that how the timing and control circuit can be designed with respects to these three instructions that is add R_1, MOV R_1, R_2 and MOV accumulator, memory.

(Refer Slide Time: 00:02:51 min)



ADD R1 : $ACC \leftarrow ACC + R1$

T_3 1: $DR \leftarrow R1$

T_4 2: $ACC \leftarrow ALU_{\underline{ADD}}(ACC, DR)$; T_0

MOV R_1, R_2

T3 : $R_1 \leftarrow R_2$: T_0

MOV ACC, M

T_0 :

T_1 :

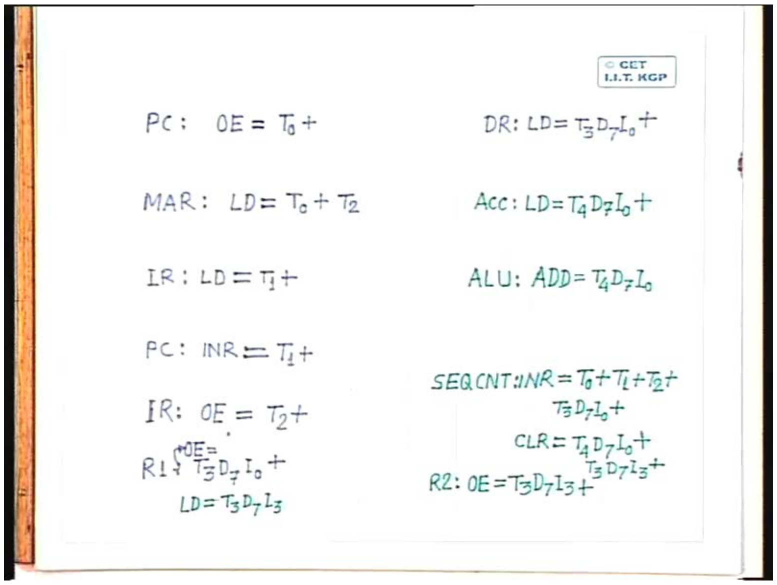
T_2 :

T3: $ACC \leftarrow M[MAR]$: T_0

So till last class we have considered only the first instruction that is the add instruction and we have said that for execution of any of the instructions, few operations like opcode fetch that is common and the opcode fetch and the decoding that takes place during the time states machine states T_0, T_1 and T_2. So for execution of any instruction, the operation during T_0, T_1 and T_2 will remain the same. So those are the common micro operations which are to be there for execution of any of the instructions. After that T_3 onwards the operations, the micro operations are different for instructions. So for add R_1 during T_3 the operations that are to be performed is transferring the content of R_1 to data register because as per our architecture, CPU architecture addition directly on R_1 is not possible because R_1 does not provide any input to the ALU.

ALU gets input from the data register. So for addition operation add R_1, the data from R_1 has to be transferred to the data register after that the content of data register can be added with the accumulator and after additional operation is complete, the accumulator output has to be loaded back into the accumulator. So this is the operation that will be performed while execution of add R_1 instruction. We were trying to develop the control logic for performing these operations.

(Refer Slide Time: 00:04:51 min)



© CET I.I.T. KGP

PC: $OE = T_0 +$

MAR: $LD = T_0 + T_2$

IR: $LD = T_1 +$

PC: $INR = T_1 +$

IR: $OE = T_2 +$

R1 $\begin{cases} OE = T_3 D_7 I_0 + \\ LD = T_3 D_7 I_3 \end{cases}$

DR: $LD = T_3 D_7 I_0 +$

Acc: $LD = T_4 D_7 I_0 +$

ALU: $ADD = T_4 D_7 I_0$

SEQ CNT: $INR = T_0 + T_1 + T_2 + T_3 D_7 I_0 +$

$CLR = T_4 D_7 I_0 + T_3 D_7 I_3 +$

R2: $OE = T_3 D_7 I_3 +$

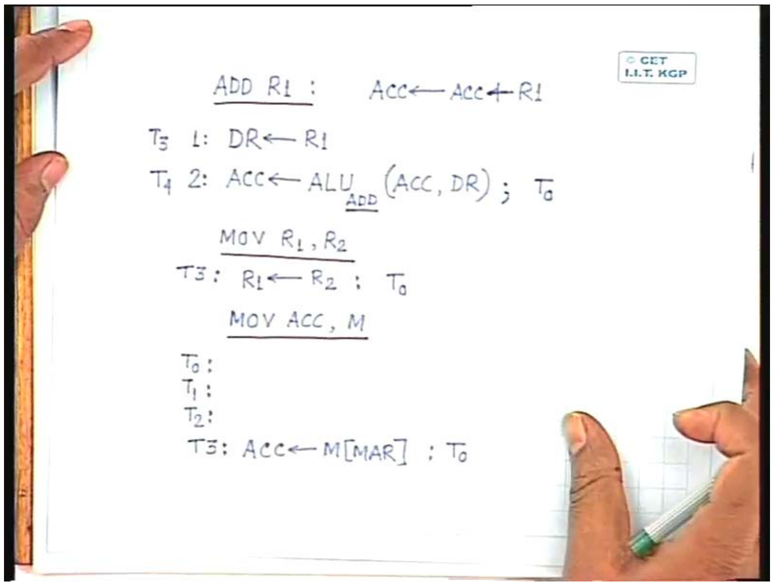
So the control logic we are developing or something like this. We have said that during T_0, T_1 and T_2, the operations for all the instructions are common. So during T_0 the operation was content of the program counter has to go to the memory address register. So output enable of the program counter, it has to be activated during machines state T_0. Then memory address register that has to get the input during T_0 and also during T_2, when the operand address part of the instruction that is IR_{0-11} the lower 12 bits will be transferred from the instruction register to memory address register. So during T_2 again the data has to be loaded into memory address register, so the load control signal till now gets the logic of T_0 plus T_2. Instruction register that instruction code is loaded into instruction register during T_1.

So the instruction register load input has to be active during T_1 and the instruction register output enable also has to be active during machine state T_2 because during that time, the operand address part of the instruction will be loaded into the memory address register. So you find that load input of the memory address register that is active during T_2. Similarly the output enable of the instruction register that is also enabled during T_2. So because these two are activated simultaneously, the output from the instruction register the lower 12 bits will be loaded into memory address register.

Then for performing the add operation add R_1, we have said that because this is a register reference instructions so the decoder output, instruction decoder output D_7 will be high. So if D_7 is high then during time interval T_3, what you have to do is we have to transfer the data from the register R_1 to the data register. So that is why if D_7 is high, referring that it is a register reference instruction and I_0 that is the least significant bit of the instruction register, we have said that for register reference instructions the operand fields identify that what register reference instruction it is. We have said that if I_0 is high then it is add R_1 operation. So during T_3, if D_7 is high and I_0 is high then data will be transferred from register R_1 to data register. So the output enable of R_1 must be active, if this condition is true. Similarly the load input of the data register also must be active if the same condition is true.

So the same logic goes to the load input of the data register. It also goes to the output enable of the register R_1.

(Refer Slide Time: 00:08:07 min)



ADD R1 : $ACC \leftarrow ACC + R1$

T_3 1: $DR \leftarrow R1$

T_4 2: $ACC \leftarrow ALU_{ADD}(ACC, DR)$; T_0

MOV R_1, R_2

T_3 : $R_1 \leftarrow R_2$: T_0

MOV ACC, M

T_0 :

T_1 :

T_2 :

T_3 : $ACC \leftarrow M[MAR]$: T_0

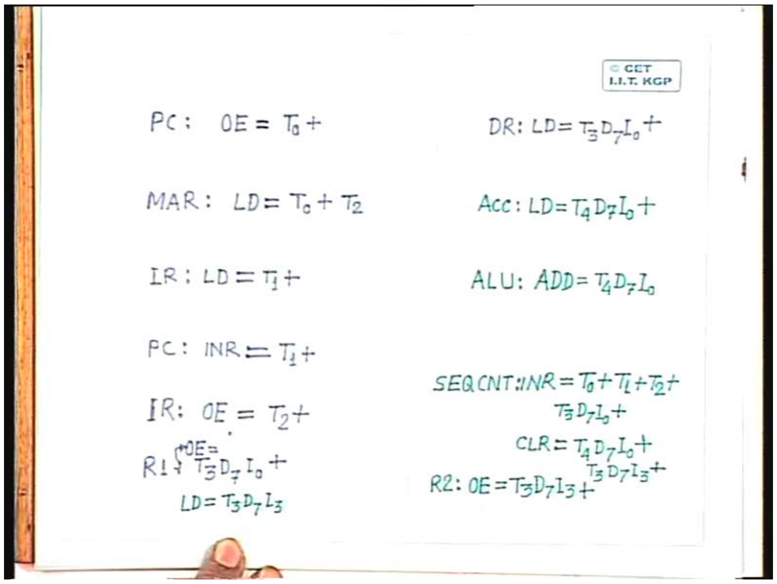
Now in addition to this for performing the addition operation, you find that during time interval T_4, accumulator will get the output of the ALU and because this is add operation, so ALU has to perform the addition operation. Among the arithmetical logical operation, we have assumed that we have only two operations. One is ADD, other one is AND. The other operation that has to be performed on the accumulator is the complementation of the accumulator for which we will assume that ALU will not be needed, q bar output of the accumulator can be fed back to the d input of the accumulator. So by using a single clock pulse, within the same machine state, the accumulator complementation can be performed.

So ALU will be involved for performing only two operations, one is ADD operation and the another one is AND operation. So in the simplest case, we will assume that ALU will also have two mode select inputs. One corresponding to ADD, the other one corresponding to AND. So whenever the ALU has to perform the add operation, the add mode select input of the ALU will be active. When it performs AND operation, the AND mode select input of the ALU will be active. So accordingly, for performing this ADD operation coming to the control signal needed for the accumulator, what we need is accumulator has to have a load input. So for the accumulator will have a load input because the result after addition will be loaded into the accumulator and that is to be performed during the time state T_4. So this load input of the accumulator will be active during T_4 when D_7 is active because this is register reference operation and I_0 is high. So that is an add operation. ALU load input may be active in other cases as well. So I will put it as OR logic. Similarly for ALU I have to have the mode select inputs. I assume that for add operation I have the mode selection input called add. This also has to be active during T_4 if D_7 is high and I_0 is high. So with the help of these control signals I can perform the add operation.

Now there is one more operation that is quite obvious that is at the end of each of these time periods machine states, the sequence counter has to be incremented by one. So for that we have an increment input of the sequence counter. So the other unit that is involved in the timing and control is sequence counter. sequence counter is having an increment output. When this increment output will be active, it has to be active during T_0 because after T_0 the machine state has to go to T_1. It also has to be active during T_1 because after T_2, the machine will go to T_2. It also has to be active during T_2 because after T_2, the machine has to go to T_3. It also has to be active during T_3 because after T_3 it will go to T_4.

Now let me do one thing. Let me not put T_3 right now. We will come to that later, for the time being let it be here. The sequence counter also has a control input called clear. So after completion of the execution of any instruction, the machine has to go back to time state T_0 when it will be ready for fetching the next instruction. So for that the clear input has to be active. Now with respect to this add operation, you find that at the end of machine state T_4, we have to bring back the machine state to T_0. So this clear input has to be active, if during T_4 if D_7 is high and I_0 is also high. It will be cleared in other situations also, so we will put this as OR logic. So with the help of this, whatever is the control signal required for performing add operation that control logic has been done. After doing this let us take the next instruction that is MOV R_1, R_2. When R_1, content of R_2 has to be transferred to register R_1 and you have seen that this operation can be performed in only one machine state that is during time state T_3 and the control signals that is required for performing this operation is output enable of R_2 and load input of R_1.

(Refer Slide Time: 00:14:32 min)



© CET I.I.T. KGP

PC: $OE = T_0 +$

MAR: $LD = T_0 + T_2$

IR: $LD = T_1 +$

PC: $INR = T_1 +$

IR: $OE = T_2 +$

R1: $OE = T_3 D_7 I_0 +$
$LD = T_3 D_7 I_3$

DR: $LD = T_3 D_7 I_0 +$

Acc: $LD = T_4 D_7 I_0 +$

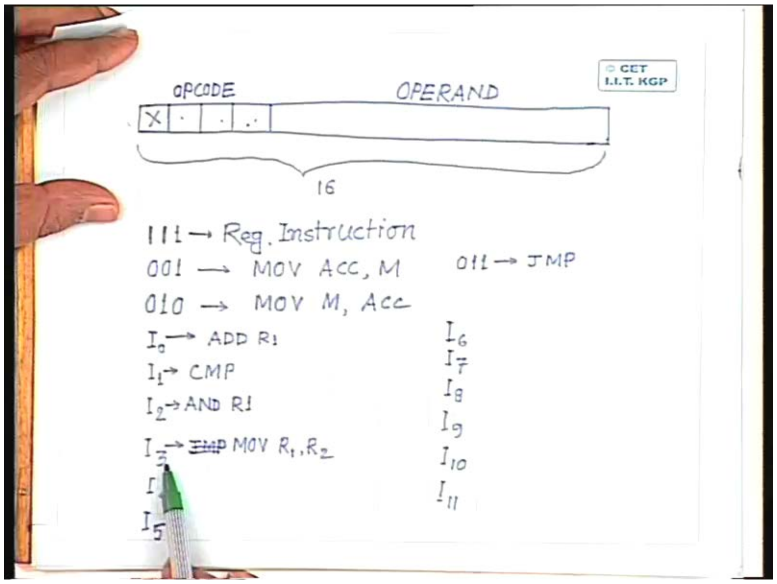
ALU: $ADD = T_4 D_7 I_0$

SEQCNT: $INR = T_0 + T_1 + T_2 + T_3 D_7 I_0 +$
$CLR = T_4 D_7 I_0 + T_3 D_7 I_3 +$

R2: $OE = T_3 D_7 I_3 +$

So we take register R_2. We take the output enable of register R_2 and for MOV R_1, R_2 what is the code? Let us see.

(Refer Slide Time: 00:15:07 min)


OPCODE OPERAND

$\underbrace{\boxed{X}\boxed{\cdot}\boxed{\cdot}\boxed{\cdot}\boxed{\qquad\qquad\qquad}}_{16}$

$111 \rightarrow$ Reg. Instruction

$001 \rightarrow$ MOV Acc, M $\quad 011 \rightarrow$ JMP

$010 \rightarrow$ MOV M, Acc

$I_0 \rightarrow$ ADD R1 $\quad I_6$

$I_1 \rightarrow$ CMP $\quad I_7$

$I_2 \rightarrow$ AND R1 $\quad I_8$

$I_3 \rightarrow$ ~~JMP~~ MOV R_1, R_2 $\quad I_9$

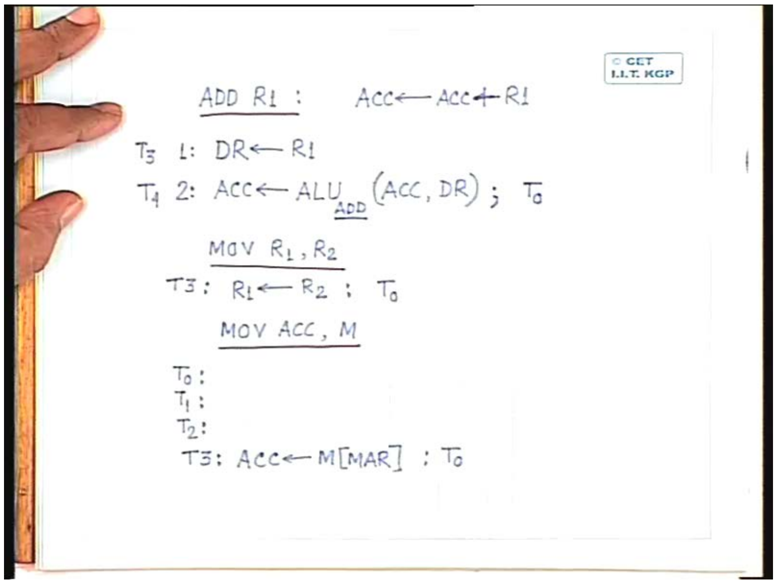
I $\quad I_{10}$

I_5 $\quad I_{11}$

This is what we have assumed for MOV R_1, R_2. That means it is also a register reference instruction. All the operations had been performed within the register and the I_3 bit of the instruction register has to be high. That means for this instruction, we have to have the opcode as 1 1 1 that means D_7 output will be high and the bit I_3 will also be high. So for output enable for register R_2, we must have this condition to be true. During T_3, D_7 is high and I_3 is also high. During the same interval, the load input of register R_1 also has to be active because the data has to go from register R_2 to register R_1. So for that what we need is the load input of register R_1. R_1 have we considered till now? No. So for register R_1, we have to consider the load input. So load input also be has to be active, if this condition is true; load of register R_1. So we have to have T_3 D_7 and I_3.

So if we set this conditions true then the data will be transferred from register R_2 to register R_1. Now in addition to this, what are the other things that we have to consider? For the sequence counter, the clear input also has to be active. If this condition is true because after T_3, the sequence counter has to generate machine state T_0. So for the clear input, we must have T_3 D_7 and I_3. So clear becomes T_4 D_7 I_0 or T_3 D_7 I_3. Now here you find that for sequence counter, I put a question mark that whether we should put T_3 also here or not? In this case after T_3 the sequence counter has to be cleared, it is not to be incremented to T_4. So I have to set what will be the increment logic for the sequence counter. So the sequence counter will be incremented after T_3 for an add operation. So for that the was the logic? T_3, D_7 and I_0, so I will put it as T_3 D_7 and I_0. Again the situation may arise in other cases. So I will put all of them are OR logic. So with this I can complete the execution of MOV R_1, R_2 statement.

(Refer Slide Time: 00:18:56 min)



ADD R1 : $ACC \leftarrow ACC + R1$

T_3 1: $DR \leftarrow R1$

T_4 2: $ACC \leftarrow ALU_{\underline{ADD}}(ACC, DR)$; T_0

MOV R_1, R_2

T3 : $R_1 \leftarrow R_2$: T_0

MOV ACC, M

T_0 :

T_1 :

T_2 :

T3: $ACC \leftarrow M[MAR]$: T_0

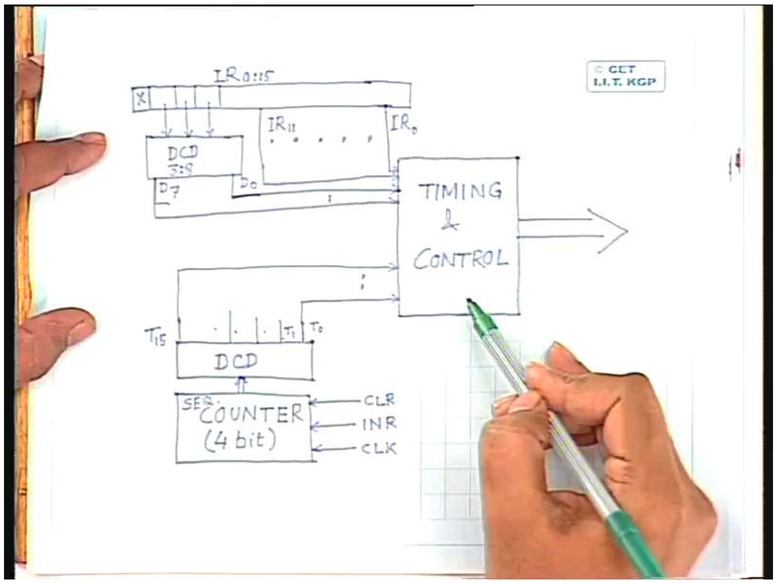
Coming to the next statement that we are considering, that is MOV accumulator, M that is reading the data from a location in memory and loading that data into accumulator. So for that we need memory address from which location in the memory that data has to be read and you see that during time interval T_2, we have already put the operand address into memory address register and this operand address has come from the instruction register. So whatever location that has to be read and the data has to be put into the accumulator, the address of that location is already available in the memory address register. So we don't have to perform any extra operation for that purpose.

So what we have to do is we have to simply generate the memory read control signal and we have to generate the accumulator load control signal during time interval T_3, if it is a memory read operation. Whether it is memory read operation or not that will load from the decoder output, from the instruction decoder output. So for this our instruction decoder output was 0 0 1 which is the instruction for MOV accumulator, memory. So our logic will be that during machine state T_3, if the decoder output one is active then what operations we have to perform? We have to generate the memory read control signal, we have to generate or activate the load control signal of the accumulator. So I will consider this. I will put, it is a memory M. For m I need the control signal memory read, I will put it as a MR. So MR will be active, if during time state T_3 if D_1 is high.

Similarly for the accumulator, the load input will be active, the load control signal will be active during T_3 if D_1 is high. I can also have other conditions so let us put this as OR logic. So this completes our memory read operation. Now again in memory read operation, after time state T_3 the sequence counter has to be cleared. So for clearing sequence counter, here I have to have the same logic that is T_3 D_1. I can have other condition as well, so I put this as OR logic. So with this I can transfer the content of the addressed memory location to accumulator and after that transfer is complete, the sequence counter will be cleared to zero, generating the next machine state as T_0, when it is ready for fetching the next instruction.

Now this memory read also has to be active for instruction fetch and that has to be done when we have put the instruction opcode into the instruction register during time state T_1. So irrespective of the instruction during time state T_1, we have to generate the memory read control signal because this is also reading a location of the memory and loading the content into the instruction register. So memory read also has to be active during time state T_1 and there may be other conditions as well. So you find that following this kind of logic after analyzing each and every instruction that what are the micro operations that are involved in the instruction and in which sequence the micro operations are to be performed, I can generate the control signals accordingly.

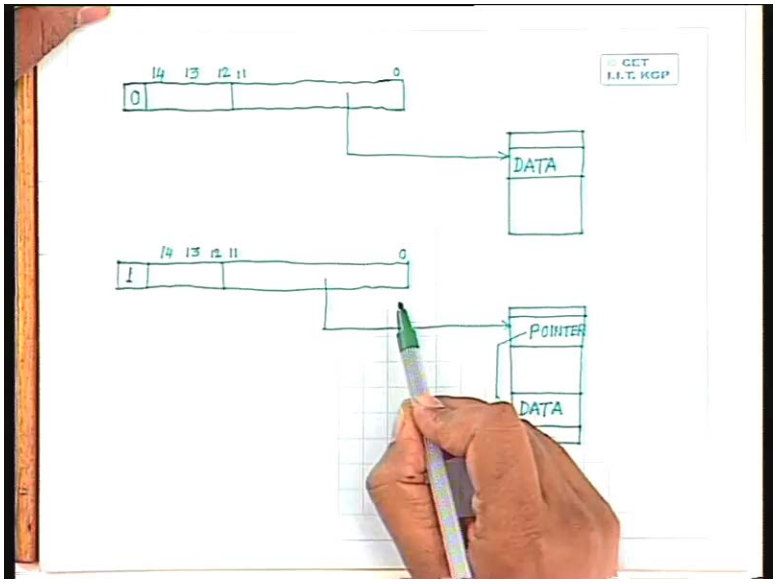
(Refer Slide Time: 00:24:00 min)



So once I get the logic for each of this control signals, I simply put this logic design a combinational circuit to replace this block with the combinational logic circuit and that combinational logic circuit comes from this logical expressions. So with that I can complete this timing and control circuit design. Sequence counter increment has to be done, why? After T_3 D_1, you are resetting the sequence counter so that comes in clear input. You cannot put both clear and increment active simultaneously because this transfer of data from the memory to the accumulator is complete during T_3. So following T_3 the machine should be ready to get the next instruction from the memory. So for that I have to generate the machine states T_0. So that is what has been done by this clear. Is that okay?

Now we will see that in this instruction register, one bit we had left in our last class. Instead of having opcode as four bit opcode, we have said that our opcodes are 3 bit up codes, one bit I had left. So what I can do is these bit I can use to indicate whether the memory reference that is being performed is direct memory access or indirect memory access.

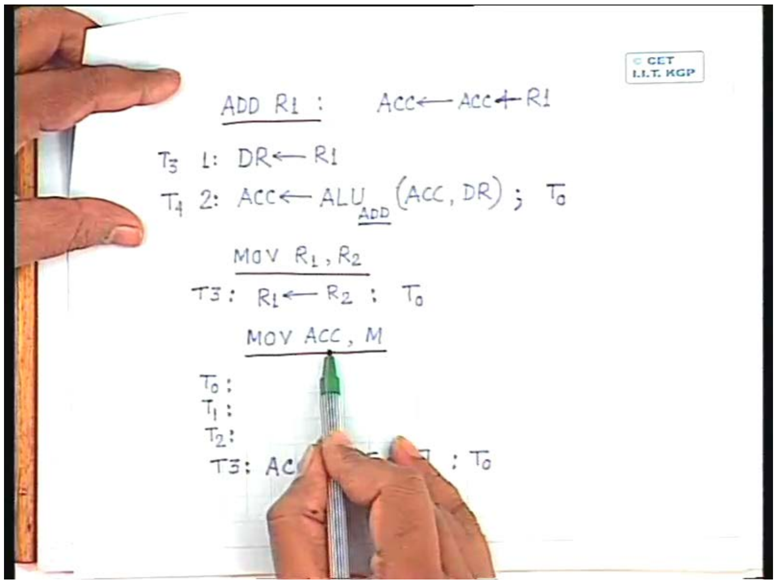
(Refer Slide Time: 00:25:44 min)



Now what is meant by direct memory access? In case of direct memory access, we have said that whatever is there as the operand address, the bits 0 to 11 in the instruction register, this gives you the operand address and for direct memory access we have said that this operand address directly specifies the location in the memory that contains the operand. So this points to a location in the memory that contains the data and for this, the most significant bit in the instruction register will have a value zero. Now for indirect address, what we will assume is for any memory reference operation, these bits from 0 to 11 which otherwise gives you the address of the operand in memory, now this is pointing to a memory location something like this. This is pointing to a memory location, this memory location gives you the address of the data. This is what contains the data.

So in case of direct address, what we have said is for a memory reference instruction, we have to have three bit opcode that is bit number 12, 13 and 14. This 3 bit opcode and the last bit indicates whether it is direct addressing or it is indirect addressing. So if the most significant bit is 0, we say that it is a direct addressing mode. In case of direct addressing mode, whatever you have in the bit numbers 0 to 11 that is this lower 12 bits of the instruction that gives you the address of the operand. That means this is pointing to a particular memory location and content of that memory location is the data.

(Refer Slide Time: 00:28:51 min)



$\underline{\text{ADD R1 :}}$ $ACC \leftarrow ACC + R1$

T_3 1: $DR \leftarrow R1$

T_4 2: $ACC \leftarrow ALU_{\underline{ADD}}(ACC, DR)$; T_0

$\underline{\text{MOV } R_1, R_2}$

T_3 : $R_1 \leftarrow R_2$: T_0

$\underline{\text{MOV ACC, M}}$

T_0 :

T_1 :

T_2 :

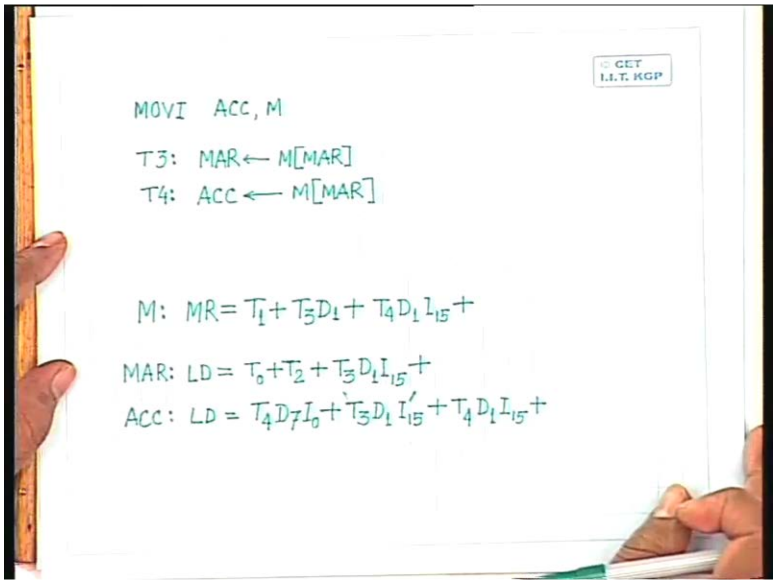
T_3 : AC : T_0

That is what we have done for this example when we have moved, executed the instruction MOV accumulator, memory here. So in this we have assumed that whatever is the content of those lower 12 bits, that is the address of the data and that is why we have during T_3 we have transferred the content of memory pointed to the memory address register to the accumulator because this is the data. Here our assumption is the most significant bit is 0. So the control logic that we have performed, in the control logic we have to put one more component that is the MSB of the instruction register has to be 0. So instruction register I_{15} complement has to come in to that.

Now in case of indirect addressing again this bit numbers 12, 13 and 14, they give you the opcode of the instruction. Bit number 15 is one. So if bit number 15 is one and this is a memory reference instruction then the addressing mode that is being used is an indirect addressing, not a direct addressing. So for indirect addressing whatever you have in this bits that is bit number 0 to 11, they point to a particular memory location. Now this memory location is not a data but this is a pointer that means it is another address. So whatever the content of this memory location, this is the address of a memory location that contains the data. So what should our control logic do now?

Control logic has to find out that if this is the memory reference instruction then it has to check whether this most significant bit is 0 or not. If the most significant bit is 0 then whatever control logic we have developed till now that is valid. If this is 1 then what we have developed is no more valid because we have to take care of this indirection. If this bit is 1, MSB is 1 and it's a memory reference instruction then I have to get the data from this location and address of this comes from this location. So during time interval T_3, during the machine state T_3 which otherwise in case of machine state T_3, what we have done is we have transferred the content of the memory addressed by the memory address register to the accumulator. If this bit is one then what we have to do is we have to read the same content but this cannot be put into the accumulator because this is not the data. The content of this we have to put to the memory address register. After putting this to memory address register, we have to perform one more memory read operation and for this next memory read operation, whatever you get from the memory that can be put into the accumulator.

(Refer Slide Time: 00:32:36 min)



MOVI ACC, M

T3: MAR $\leftarrow$ M[MAR]

T4: ACC $\leftarrow$ M[MAR]

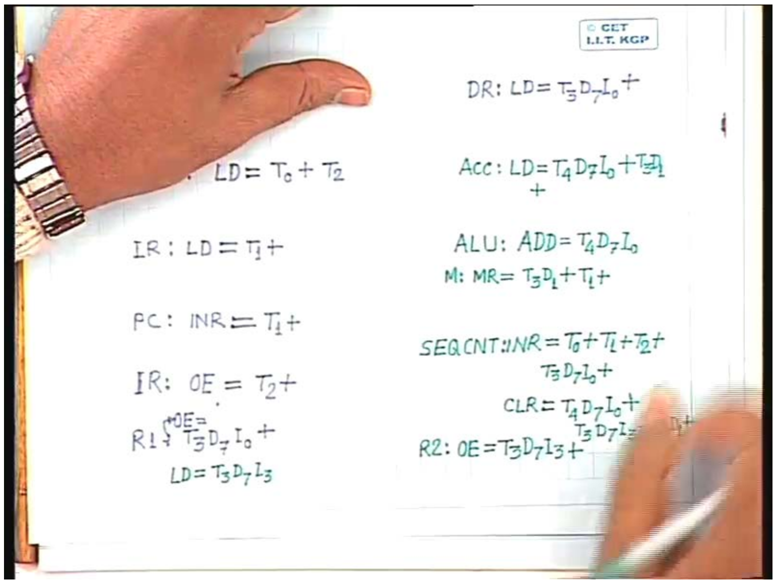
M: $MR = T_1 + T_3 D_1 + T_4 D_1 I_{15} +$

MAR: $LD = T_0 + T_2 + T_3 D_1 I_{15} +$

ACC: $LD = T_4 D_7 I_0 + T_3 D_1 I'_{15} + T_4 D_1 I_{15} +$

So for this indirect memory transfer, let me put it as MOVI. The operands remains the same accumulator, M. So what is the additional operation that we have to perform? During machine state T_3, previously we have read the memory and put the data into the accumulator. Now what we have to do is again you read the memory, address coming from the memory address register instead of putting this into accumulator, we have to put this into memory address register itself. So I assume that I have that corresponding bit mapping. So during T_3, this is the operation that have to perform. During T_4, I have to perform another memory read operation and now the data will come to the accumulator. It will not go to the memory address register any more. What is the content of memory address register now? Memory address register, that is the pointer that is the address that has been set during T_3. So for this our control logic has to be suitably modified.

(Refer Slide Time: 00:34:37 min)



© CET I.I.T. KGP

$LD = T_0 + T_2$

IR : $LD = T_1 +$

PC : $INR = T_1 +$

IR : $OE = T_2 +$

R1 : $OE = T_3 D_7 I_0 +$

$LD = T_3 D_7 I_3$

DR : $LD = T_3 D_7 I_0 +$

Acc : $LD = T_4 D_7 I_0 + T_3$ [illegible] $+$

ALU : $ADD = T_4 D_7 I_0$

M : $MR = T_3 D_1 + T_1 +$

SEQ CNT : $INR = T_0 + T_1 + T_2 +$

$T_3 D_7 I_0 +$

$CLR = T_4 D_7 I_0 +$

$T_3 D_7 I_3$ [illegible]

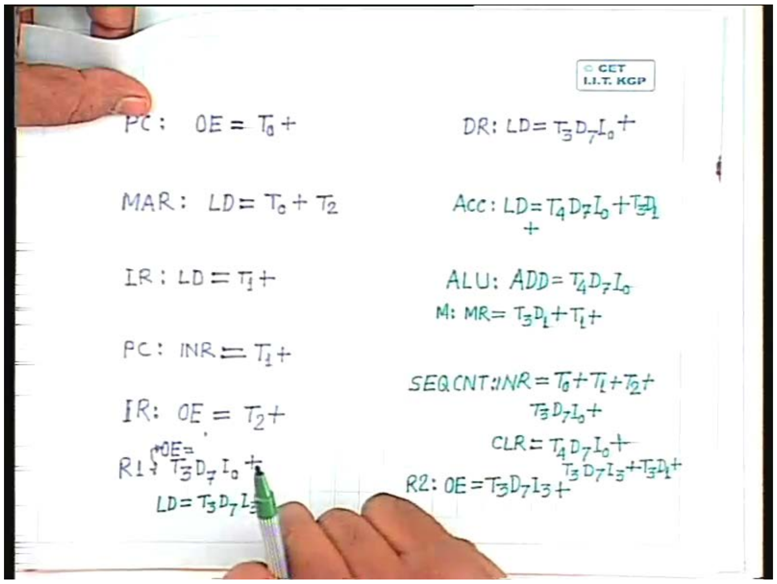
R2 : $OE = T_3 D_7 I_3 +$

So in earlier case for memory read, we have assumed that during T_3 if D_1 is high then we to perform, we have to enable the memory read control signal. Memory read control signal will, in any case be activated during T_1 because that is an opcode fetch operation. So it is respective of the instruction that we are going to execute. So now again similarly for memory read, for the memory how do you activate the memory read control signal? Now memory read control signal will be during T_1, we have already activated this memory read plus during T_3 if D_1 is high then we have to perform memory read.

Now we find that during T_3 whether it is direct or indirect, in both the cases we have to read the content of the memory, only the destination will be different. So I simply put as T_3 D_1. I don't check what is the content of the most significant bit in the instruction register. I have to perform an additional memory read operation during T_4, if the most significant bit of the instruction register is one. So I will put it as T_4 and if the opcode is D_1, not only that if I_{15} the most significant bit of the instruction register is one then also I have to perform memory read operation. So this is the additional memory read operation that is required because of introduction.

I may have other logics as well, so put this as OR logic. For memory address register, I have to modify the control logic. The earlier control logic was this. During T_0 and T_2, so they will remain as it is. So for memory address register the load input, the earlier conditions remains valid during T_0 and also during T_2. We have to activate the load input of the memory address register. Now additional thing comes here. So during T_3 I again have to activate the load input of the memory address register if it is an indirect memory address operation. So my condition will be during T_3, if D_1 is high and I_{15} is high. Again I put this as a OR logic because there may be other conditions as well during which this memory address register has to be active.

(Refer Slide Time: 00:38:05 min)



© CET I.I.T. KGP

PC : $OE = T_0 +$

MAR : $LD = T_0 + T_2$

IR : $LD = T_1 +$

PC : $INR = T_1 +$

IR : $OE = T_2 +$

R1 : $OE = T_3 D_7 I_0 +$

$LD = T_3 D_7 I_3$

DR : $LD = T_3 D_7 I_0 +$

Acc : $LD = T_4 D_7 I_0 + T_3 D_1 +$

ALU : $ADD = T_4 D_7 I_0$

M : $MR = T_3 D_1 + T_1 +$

SEQ CNT : $INR = T_0 + T_1 + T_2 + T_3 D_7 I_0 +$

$CLR = T_4 D_7 I_0 + T_3 D_7 I_3 + T_3 D_1 +$

R2 : $OE = T_3 D_7 I_3 +$

Coming back to accumulator, for accumulator the load input earlier was during T_4 if D_7 is high and I_0 is high. Now this is the one that will be modified now. So these conditions remains as it is because it's a register reference instruction and the operation was moving the data from R_2 to R_1. Sorry this was the operation for adding the content of R_1 with accumulator and loading the data back into accumulator. So that will remain as it is. So for accumulator control, the load input will be modified as, the first condition we have to retain that is T_4 D_7 and I_0. This will remain as it is.

The second condition that we had put is T_3 D_1. Now T_3 D_1 will be replaced by T_3 D_1 I_{15} complement because here the load input will be active only if it is an direct memory read operation. If it is indirect memory read operation, the data will not come to accumulator. So I will put it as T three D one and I fifteen complement.
What is the additional condition? That is during T_4, the data will be loaded into accumulator if it is D_1 and I_{15} is high. I can have additional conditions, situations when the load input of the accumulator also has to be active. So you put all of them as OR logic. So this is for getting a data from a memory location, loading the data into accumulator that is the memory read operation.

Similarly for a memory write operation, when the transferring the data from the accumulator to a particular memory location I can also have direct memory write operation. I can also have indirect memory write operation. In such case the control signals that will be generated is not memory read, I have to have another control signal which is memory write. So memory write control signal also has to be activated accordingly. Similarly this accumulator, whenever you transfer the data it's a memory write operation, you have to transfer the data from the accumulator to a location in the memory. So we have to accurate the output enable of the accumulators accordingly. So this is how by analyzing the micro operations that will be performed while execution of every instruction. To that analysis I have to find out that what will be the logic for each and every control signal and I have to design the circuit for generating these logics and that has to be replaced in the timing and control circuit block.

Now here you find that though the circuit is a combinational circuit but because of the sequence counter output is also going to the timing and control circuit input, the sequences in which the control signals will be generated that is defined, because in the timing and control circuit block we have these inputs either T_1, T_2, T_3 three. So these are the inputs which are going to that particular block. So these input guarantee that these control signals will be generated during a particular machine state only. This control signals will not be generated arbitrarily.

So I can design the timing and control circuit for a CPU, given the instructions and having the knowledge of what the instructions are supposed to do. So far we have considered for the internal data path that all the components are capable of loading the data from the internal data path. They are also capable of sending the data to internal data path from there they can go the destination. For that purpose, for each of the components we have an output enable. So our restriction is the output enable of only one component can be active at a time. I cannot activate the output enable of more than one component simultaneously because in that case there will be data clash on the parser which is called bus contention. So this can be guaranteed in one of the two ways. I can assume that the outputs of each and every component is tri stated output. So only when you give the output enable component, only when you activate the output enable control signal then only the data will move from that particular selected component to the data bus. When the output enable control signal is low in that case, the output of that particular component will go the tri-state, the third state that means high impedance state and the data from that component will not reach the data bus. So that way we can avoid the parse contention.

The second approach is let there be number of components. The outputs of all these components goes to a multiplexer and it is the output of the multiplexer that acts as a parse. So now in this case I don’t need tri stated output device, I can have multiplexer. So this output enable signal now will be replaced by multiplexes select channel signal. So assuming if the accumulator is connected to multiplexer input to 0, so whenever the accumulator output is to be activated, instead of activating the output enable of the multiplexer I will properly select the select inputs instead of activating the output enable of the accumulator, I will properly select the select inputs of the multiplexer. So that only the accumulator output goes to the internal data path. So I can have one of the two ways either we can use a multiplexer or we can have tri stated outputs for every individual component in a CPU. So I hope with this discussion now you know that given any CPU and the instruction set of the CPU, you can design that particular CPU. So with this our low level design is complete. Next class onwards everything will be in block diagrams.

(Refer Slide Time: 00:46:12 min)

Preview of Next Lecture

(Refer Slide Time: 00:46:15 min)

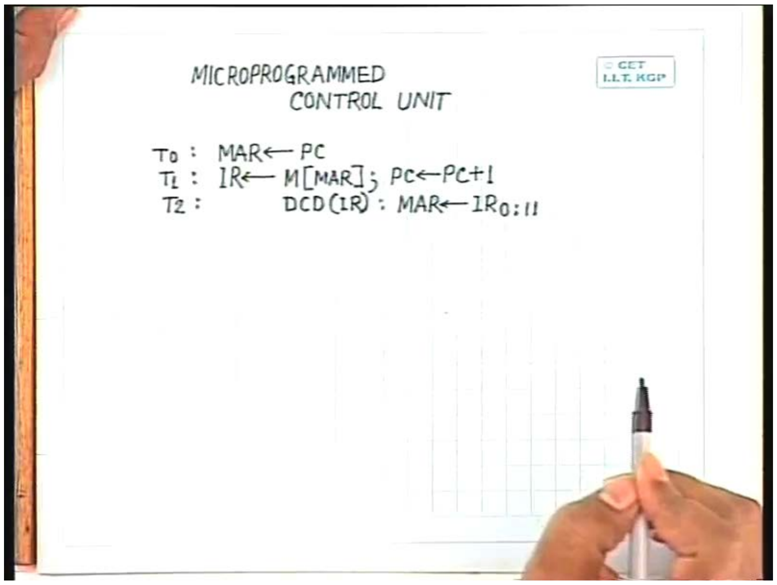
Lecture # 05

Microprogrammed Control

Till last class the control circuit unit that we have discussed that is called harder control unit because all the control signals are generated by harder circuit. Now there is another way of generating the control signals and that is called a micro programmed control unit. So in case of micro programmed control unit, the control signals instead of being generated by a hardware circuit it is generated through software and because these control signals are generated through software this is more flexible. So now let us see how you can generate the control signals through software.

So you have said during our previous discussion that once you design the hardware resources within the CPU, for every hardware resource you can determine that what are the control signals that will be required.

(Refer Slide Time: 00:47:26 min)



MICROPROGRAMMED CONTROL UNIT

T_0 : $MAR \leftarrow PC$

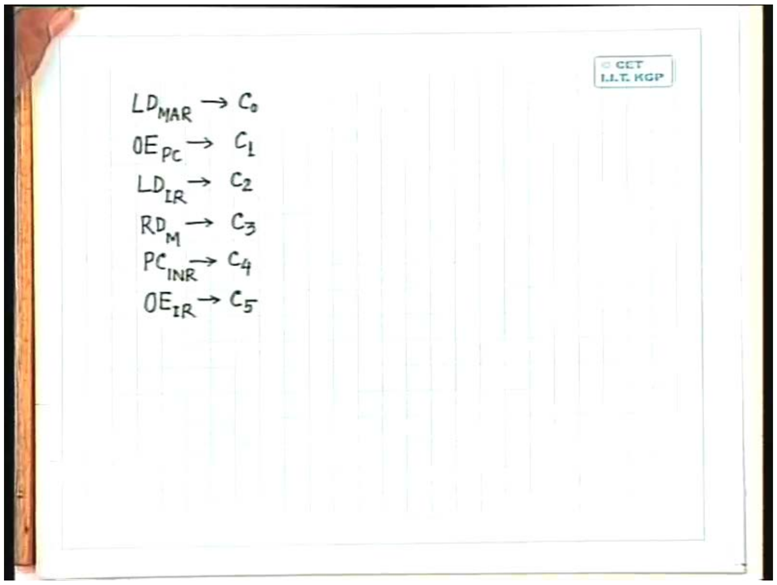
T_1 : $IR \leftarrow M[MAR]$; $PC \leftarrow PC+1$

T_2 : $DCD(IR)$: $MAR \leftarrow IR_{0:11}$

So you know our hardware control unit when we have discussed, we have said that during time state T_0 the operation that was preformed was the memory address register gets the content of the program counter. Then during time interval T_1 or machine set T_1, what we have done is instruction register gets the value from memory whose address is in the memory address register and at the same time, the program counter is incremented by one and during T_2 the content of the instruction register is decoded. So what we do is we decode the instruction register content.

So these of the operations that were done during the machine states T_0 to T_2 and we have also said that simultaneously what we have done is we have loaded memory address register with the lower 12 bits of the instruction register that is supposed to hold the operand address in memory, if it is memory reference instruction. So if I consider only these three machine states, you find that the controls that are invoked is for memory address register we have to have the load control input of the memory address register, output enable of the program counter, load instruction register, read memory, increment program counter, load memory address register again, these are identical then output of enable of the instruction register. So let us associate these control signals with some bits in the control memory.

(Refer Slide Time: 00:50:02 min)



$LD_{MAR} \rightarrow C_0$

$OE_{PC} \rightarrow C_1$

$LD_{IR} \rightarrow C_2$

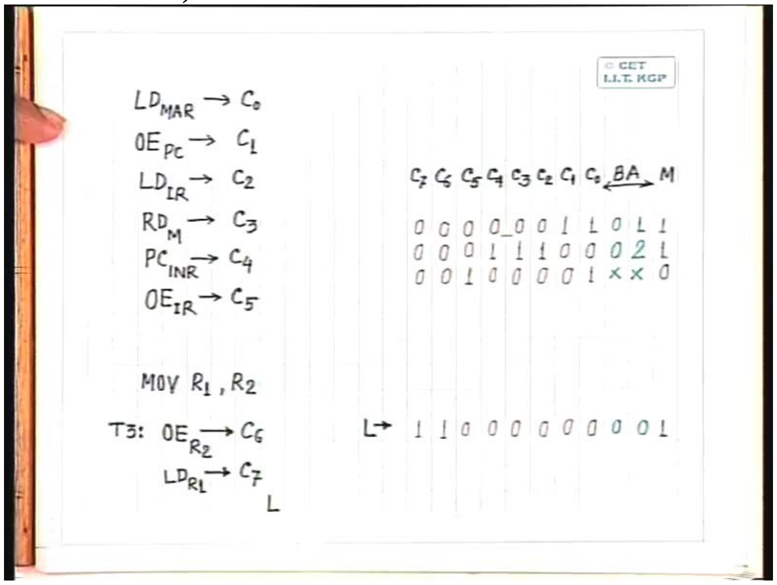
$RD_M \rightarrow C_3$

$PC_{INR} \rightarrow C_4$

$OE_{IR} \rightarrow C_5$

So let me say that load memory address register that is associated with number C_0 that is the zeroth bit in the control memory. Output enable of the program control that is associated with bit number one in the control memory or C_1, then load instruction register is associated with bit number C_2 in the control memory. Then rate control single for memory is associated with C_3 in the control memory. Then we have program counter increment that is associated with bit number C_4 in the control memory. Then load memory address register is already considered then what we need is output enable of instruction register is associated with bit number C_5 in the control memory. The first operation that is to be performed is an opcode fetch. So if I ensure that whenever the machine is powered on or whenever the machine is reset, the control memory address register will also be reset to zero.

(Refer Slide Time: 00:51:47 min)



$LD_{MAR} \rightarrow C_0$

$OE_{PC} \rightarrow C_1$

$LD_{IR} \rightarrow C_2$

$RD_M \rightarrow C_3$

$PC_{INR} \rightarrow C_4$

$OE_{IR} \rightarrow C_5$

MOV R_1, R_2

T3: $OE_{R_2} \rightarrow C_6$

$LD_{R_1} \rightarrow C_7$

L

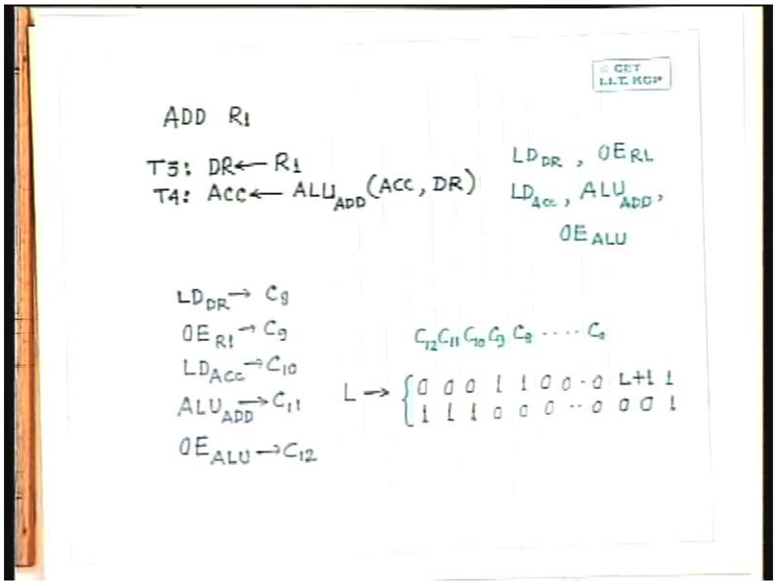
	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0	BA		M
	0	0	0	0	0	0	1	1	0	1	1
	0	0	0	1	1	1	0	0	0	2	1
	0	0	1	0	0	0	0	1	×	×	0
L→	1	1	0	0	0	0	0	0	0	0	1

So that ensures that just after switching on the machine or just after resetting the machine, the micro programmed control unit will start generating control signals from the zeroth location in the control memory. That means the first control signals it will generate are C_0 control signals associated with C_0 and C_1 which are nothing but load memory address register and output enable of the program counter. From the program counter the address goes to memory address register.

Now in addition to this we also have said that every location in the control memory will have to more fields. One is the next address field or from the next control signals are to be read and it also have a one bit field which is the modes, the address select field. So I will put in along with this in the same locations, the next address fields within the control memory address register. So you will find that after generating this control signals, the next control signals are to be generate from the next memory location within the memory, which is memory location one. So I put the next memory location in few more bits, I put it as 0 1. Right now I am putting it decimal form but this has to be coded in to binary and the number of bits required in binary form that has to be used. For simplicity I am putting this as the decimal number.

So after execution of this, the next address is one that is this particular location in the control memory. So after generations of this control signals, next time the control signals will be generate are C_4, C_3 and C_2 and C_4, C_3, C_2 means increment program counter, read memory and load instruction register. So after these control signals are generated, the next control signals are to be read from the third location in the control memory. So I again put in the next address field as 0 2, so you find that I am putting in the decimal form. So we find that for the simple situation MOV R_1, R_2 the micro program was a single micro instruction. So this I can call as a micro instruction, so it is a signal micro instruction whereas for execution of some other programs may be I need more than one micro instruction. I think we had taken some such example where you need more than one.

(Refer Slide Time: 00:54:55 min)



ADD R_1

$T5$: $DR \leftarrow R_1$ $\quad LD_{DR}$, OE_{R1}

$T4$: $ACC \leftarrow ALU_{ADD}(ACC, DR)$ $\quad LD_{Acc}$, ALU_{ADD}, OE_{ALU}

$LD_{DR} \rightarrow C_8$

$OE_{R1} \rightarrow C_9$

$LD_{Acc} \rightarrow C_{10}$

$ALU_{ADD} \rightarrow C_{11}$

$OE_{ALU} \rightarrow C_{12}$

$$\begin{array}{l} \quad\quad C_{12}\,C_{11}\,C_{10}\,C_9\,C_8 \cdots C_0 \\ L \rightarrow \begin{cases} 0\ 0\ 0\ 1\ 1\ 0\ 0 \cdot 0\ \ L+1\ \ 1 \\ 1\ 1\ 1\ 0\ 0\ 0 \cdots 0\ 0\ 0\ 1 \end{cases} \end{array}$$

The example that we had taken is add. One of the instructions that we had is add R_1. In add R_1, the micro operations that are to be done is during time machine state T_3, the content of R_1 has to be loaded to data register. Then during machine state T_4, accumulator is to be loaded with sum of accumulator and data register. So that we had put in this way ALU performing add operation on accumulator and data register. So for this operation I need a micro instruction and the control signals that are needed are load the data register and output enable of R_1. For this micro operation to be executed during machine state T_4, the control signals which are required are load accumulator then we need ALU add control signal because when this control signal is made equal to one then only ALU will perform add operation.

Simultaneously we also need output enable of ALU because from the ALU, output of the ALU is going to the common data path. So I also have to activate output enable of ALU. But does it have only advantage? Yeah, speed wise this will be slow because for generation of any control signal, I have to read the memory content. So because it is software in nature, it will be slower than hardware control unit but the advantage is it is more flexible, gives flexibly while designing and the second advantage is it is more compact because the full lot of hardware control circuit is now put in just the control memory. So it is compact and flexible but speed wise it will be slower. (Student: Refer Slide Time: 57:58). That accuracy does not matter much. So with this we will take a break.