Digital Computer Organization

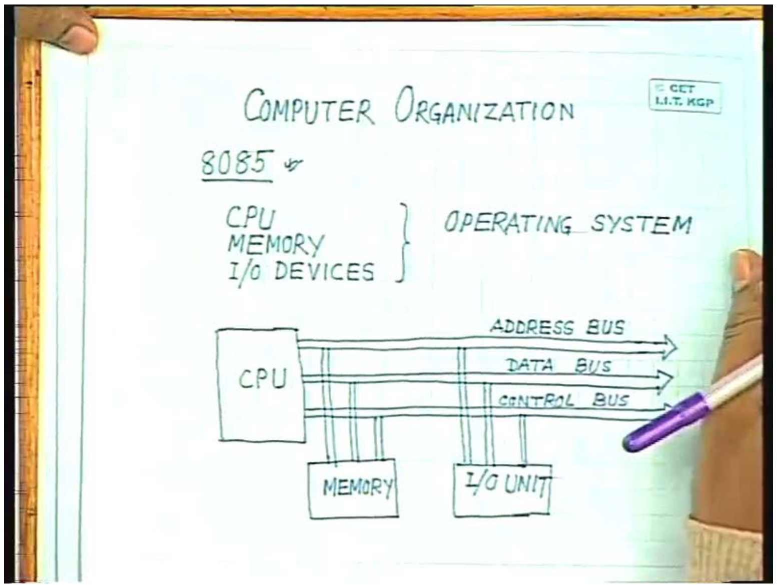
Prof. P.K.Biswas
Department of Electronic & Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture No. # 02

CPU Design - I

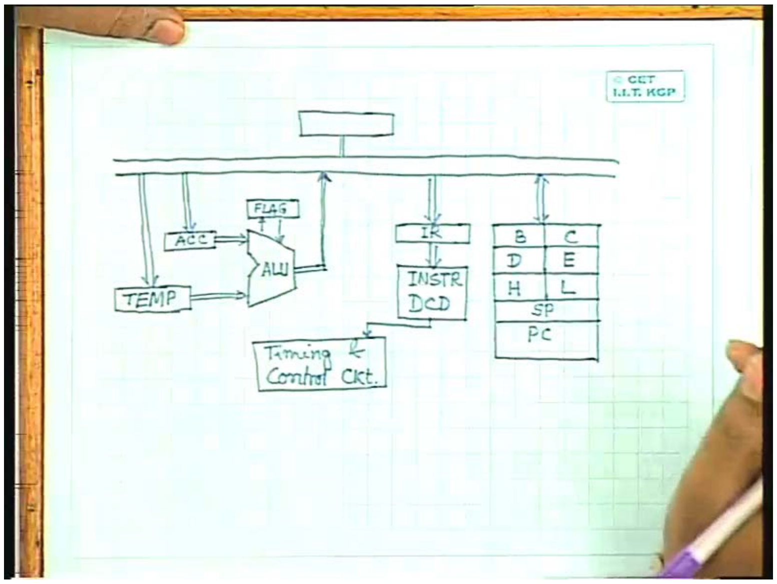
So let us start with our second lecture on computer organization. So in the last class we have seen that for a particular case of CPU that is 8085, how the interfacing is normally done.

(Refer Slide Time: 00:01:14 min)



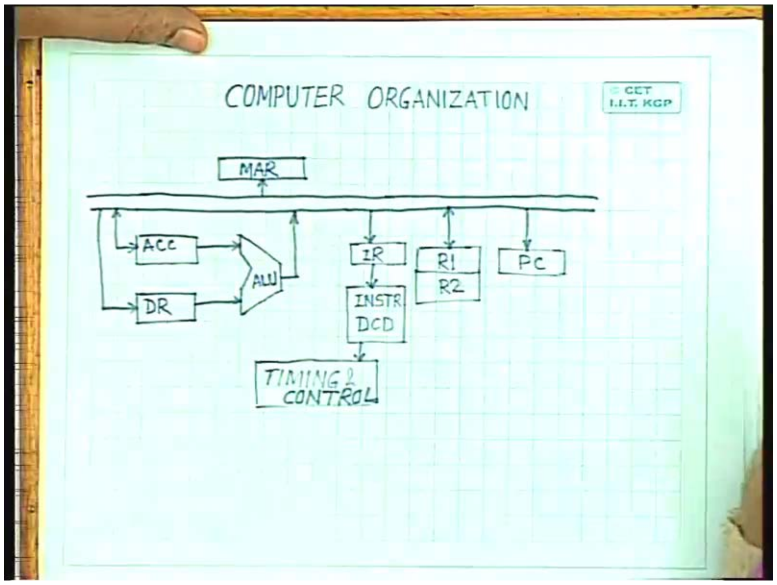
We have the CPU, we have address bus and control bus and to this address bus and control bus you have to interface the memory units and the I/O units to give a complete micro computer system. Then we had seen what is the internal architecture of 8085 CPU.

(Refer Slide Time: 00:01:37 min)



Here we have seen that internally the 8085 has got few registers like B C D E H L and a register, accumulator, temporary register and along with that you have instruction register, instruction decoder and timing and control circuit. So today we will see that we will take a simplified version of the CPU and try to design that particular CPU. So we will not take this entire 8085 but rather a subset of this, just to give you a feel about how the CPU can be designed.

(Refer Slide Time: 00:02:21 min)

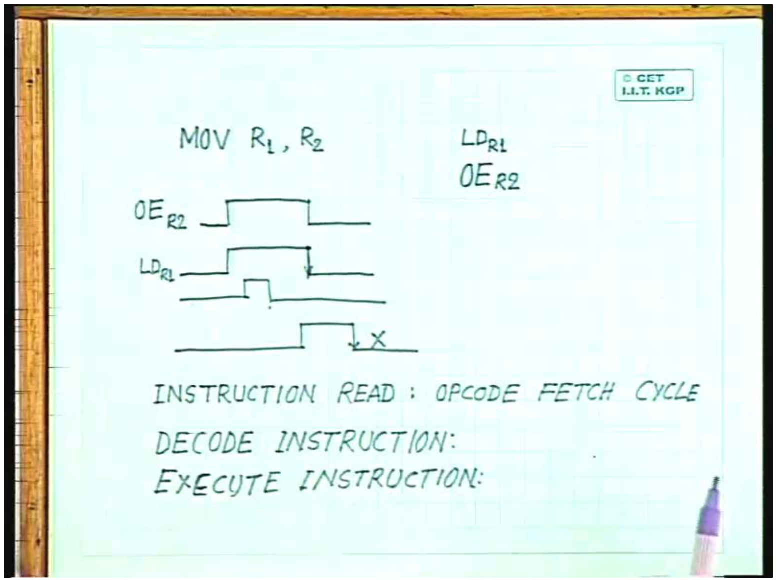


So in our design we will consider a CPU having as before obviously a CPU has to have an arithmetic and logic unit.

So we will have an ALU, it will have an accumulator and it will have a temporary register. Now let us call it a data register or DR and among other registers let us have two general purpose registers. One is let us say register R_1 and other one is register R_2. We will have a program counter. We will have, obviously we have to have an instruction register. We have to have an instruction decoder. We have to have a timing and control circuit and let us have a memory address register. In our case we will assume that this memory, the address register is a special purpose register. Unlike in case of 8085 where you have a general purpose register that is the (Refer Slide Time: 00:04:59) register pair which again acts as the memory address register but in our design we will consider that we have a separate special purpose register which is memory address register.

So our assumption is whenever some value is loaded into the memory address register that activates the external address bus. That means whenever you put some value in the memory address register, on the external address bus that value is available. So it can be externally decoded and given to the memory units or it can be externally decoded and given to the I/O units for I/O access. Then obviously we have to have an internal data path for transferring the data from one register to another register. So this is a simplified version of the CPU which we will try to develop and mainly we will try to see that what is the logic of the timing and control circuit and what are the control signals and in which sequence the control signals will be generated by this timing and control circuit. So before going into details of this CPU design, to explain that what are the control signals that will be required let us take a very simple instruction.

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



INSTRUCTION READ : OPCODE FETCH CYCLE

DECODE INSTRUCTION:

EXECUTE INSTRUCTION:

That is a simple instruction of the form say MOV R_1, R_2. As you know that such a type of assembly language instruction like MOV R_1, R_2 is meant for transferring the data from some source register to a destination register. Conventionally we follow it this way that the first register which is mentioned that is the destination and the second register that is mentioned in the instruction that is the source.

So this instruction MOV R, R_2 will perform the function of transferring the content of register R_2 to register R_1. So what are the control signals that will be required to execute this instruction. So right now let us consider only the execution part. Instruction fetch opcode decode that we will consider later on. So only for execution of this instruction, what are the control signals that are required? Obviously since you have to load the data in register R_1 which is the destination. So register R_1 must have a control signal called load or latch. We will call this control signal as load R_1. Whenever you send a pulse to this load R_1 input, load input of register R_1, whatever is available at the input of register R_1 at that instant that will be loaded into register R_1. So that kicks takes care of loading a data into register R_1.

Now this data has to come from register R_2. Now if I consider this particular architecture, you find that all the registers R_1, R_2 program counter, memory address register all of them are connected to a common data path. So which says that whenever I have to transfer a data from a source to some destination, in that case the content of the source must be available on the common data path. Not only that since I have more than one registers connected over the common data path and any of the register is capable of sending data onto the common data path. So there I must be selective. I must ensure that whenever I select register R_1 to send the data onto the common data path, no other register connected on the same data path should be able to send the data onto the data path because in that case you will have data clash. So I must have a selector. The responsibility of the selector will be to select that particular source to send the data onto the data path which is mentioned in the instruction.

So in this case I must have because here R_2 is the source, R_2 must have some control input which when activated will enable R_2 to send the data onto the data path. When it is inactive, R_2 will not send the data onto the common data path. So that particular control signal of register R_2 we will mention as output enable of R_2. So we find that for execution of this particular instruction MOV R_1, R_2 I need just these two control signals. One is output enable of R_2 by which the data of R_2 will be available onto the common data path and while it is available onto the common data path, I have to activate load input of R_1 so that the data is loaded from the common data path into register R_1. So these are the control signals which are to be activated and they have to be activated almost simultaneously because I have to ensure that when I try to load the data into R_1 during that time output enable of R_2 must be active. It should not so happen, that first to enable output of R_2, you withdraw that enable signal and then you give load input of R_1 because in that case whatever will be loaded into R_1 that is not the data that you want because before enabling load input of R_1, you have removed the output enable of R_2. That means the data of R_2 is no more available onto the data path.

So the situation should be something like this. First you enable the output of R_2, so this if I say this is a pulse which enables output of R_2. Then load input of R_1 should be within this region, it should not go beyond this. So in one situation I can have load input exactly same as output of enable of R_1. This will also do. This will also do because this load input of R_1 coincides with output enable of R_2. We can assume that the data loading will take place at the following age of this block. But when the data is actually loaded till that time I must ensure that output enable of R_2 is active. So which is true in this case that is also true in this case. But if I have a situation something like this, this is not permitted because when you are trying to load the data into register R_1, you have already removed the output enable of R_2.

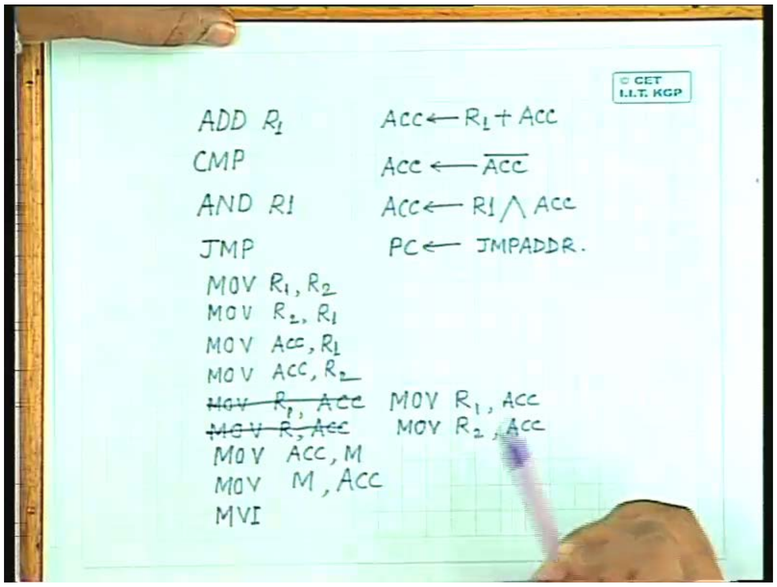
So what will be loaded into R_1 with this load signal that will be a garbage data. This is not the data which is actually needed. So when you consider the control signals and the sequence in which the control signals are to be activated, you have to keep in mind all these things that you should not load any garbage data in to any of the source. So this is just to give you a feel about what are the control signals that will be needed and in which sequence the control signals are to be activated. This takes care of only the execution part but as you know that for execution of any instruction, you have various stages. For example the instructions are always loaded into the main memory. So for execution of any instruction, firstly you have to read that instruction from the main memory, put it into instruction register then the instruction will be decoded and depending upon the decoder output, it's the timing and control unit that has to decide that what are the control signals that will be generated and in which sequence they will be checked.

So for execution of any instruction, firstly you have to have an instruction read operation which is identical with a memory rate but in this case, the difference is while execution of an instruction for a memory read operation, the data will be sent to R_1 or R_2 or accumulator depending upon the instruction type. Whereas for instruction read, whatever is read from the memory that always has to go to the instruction register. So this destination is predefined and this is also known as of opcode fetch cycle. So after instruction read, the instruction has to be decoded. So you have decode instruction first and after decoding the instruction then only the instruction will be executed.

So the next cycle is execute instruction and while execution the other things whatever is necessary that will be done. Now when you execution an instruction, the instruction can be of various kinds. For example it may be register reference instruction. For example like this MOV R_1, R_2 where you are transferring the data from register R_2 to R_1. Both the registers are internal to the CPU. So for execution of this particular instruction, I need not go to the external circuit. That is I need not have any data from the memory or I did not have any data from the I/O devices. So these are the instructions which are called register reference instructions or register instructions. There can be other types of instructions for example reading a data from the memory where the first instruction, read instruction decode will be identical but while executing the instruction, it has to invoke another memory read or memory write operation which is again similar to instruction read operation. Because here I have to again access the memory, read the data from the memory. So that read operation will be identical to instruction read but now it has to be initiated while executing the instruction.

Similarly if it is memory write operation, the data will go from some register to the memory. So again you are activating the external memory device for writing the data into some memory location. The address of the memory location will come from the memory address register. So how this control circuit, timing and control circuit will generate the timing signals and it which sequence that will be decided during the execution phase. The control signals have to be decided by what are the instructions that you have in the instruction set. So the first stage or the first step in designing a CPU is to decide that what are the instructions that you wanted to have for that particular CPU. So for this simple CPU I will assume few instructions and our design will be for those instructions only. So as I said in our introductory class that for any instruction set, some of the instructions are necessary and other instructions can be derived out of that.

(Refer Slide Time: 00:20:21 min)



ADD R_1 $Acc \leftarrow R_1 + Acc$

CMP $Acc \longleftarrow \overline{Acc}$

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, Acc~~ MOV R_2, Acc

MOV ACC, M

MOV M, ACC

MVI

So accordingly I will assume that I will have one instruction which is a add instruction. So add instruction is meant for adding two operands and I will assume that this add instruction is always a register reference instruction that means I can have add R_1. Now here you find that I am specifying only one register, only one operand not both the operands because the other operand will be the accumulator by default. So what this add R_1 do is it adds the content of the accumulator with register R_1 and saves the result in register accumulator. So this is the operation that will be performed, R_1 added with accumulator and the result goes to the accumulator. So because in this case the destination is predefined that is the accumulator I need not mention a second operand explicitly within the instruction. So simply add R_1 will do. So as you already know that will the help of this add operation, you can implement multiplication because multiplication is nothing but repeated operation.

What I cannot implement is subtraction. So for subtraction I need an additional thing that is complement. Again if I assume that this complement will always walk on one register that is the accumulator. Complement is not allowed on any other register. So I need not explicitly mention any of the registers because source and destination all of them are predefined. So this complement will perform the operation of accumulator gets the complement of the accumulator but what I have to have in this case is the accumulator hardware must support the facility of complementing itself. So if the accumulated hardware supports that just by issuing one control signal, this operation will be performed. That is very simple because as you know that registers are nothing but arrays of flip flops. So you simply feedback Q bar output to D input and activate load. Isn’t it? Just in feedback, the Q bar output to D input and activate the load in that case whatever is Q bar that will be loaded into the flip flop. That means the next Q output will be same as previous Q bar output and this complementation is done. So when I have this add along with complement, I can implement subtract and if I can implement subtract, I can also implement divide operation by repeated use of add and complement operations. So I can have addition, I can have multiplication, I can have subtraction, I can have division all these operations are possible by making use of only these two operations, add and complement.

So that takes care of my arithmetic operations. Now what about logical operations? Along with this if I simply put one logical instruction that is a AND, let me put it as And R_1. This will perform the operation of logically anding the content of register R_1 with the content of the accumulator and storing the result in the accumulator. So the content of register R_1 will be bit by bit anded with the content of the accumulator and the result will be stored into the accumulator. This is what will be done by AND operation. Compliment, it gives the compliment of the accumulator. So we find that when I have this AND and compliment together that means I can perform NAND operation and if I am able to perform NAND operation, I am able to perform any type of digital operation, logical operation because NAND function is functionally complete. So just by having these three instructions, I can perform all the arithmetic operations, I can perform all the logical operations.

Now in addition to these arithmetic logical operations, i need some more operations that is some of the control operations. So I can have one such instruction let us say jump instruction. It may be jump conditional or jump unconditional but it is better to have a jump conditional instruction because unconditional you can always do, if you always set that condition equal to true. But for this simple design I will not go into various such conditions. Let me assume that I have just a jump instruction. What are the other instructions that you need? Again compare can be implemented by subtraction. Yeah move that is called a data transfer instruction. So I have to have move instruction

What about the other registers? I have only mentioned register R_1. I have various other registers in the CPU those have to be accessed and if my instruction says that whenever I have to have an ADD operation that operation will be performed on register R_1. That is what my instruction says here. That means if there is something which is in register R_2 which is to be added to accumulator before that addition, the content of register R_2 has to be moved to register R_1 because that is the one that my instruction set permits. My instruction set does not permit addition of a number in register R_2 with accumulator. Otherwise within the instruction set, I have to include those instructions also. Isn't it? No, R_1 is not variable. Register A in 8085 is not variable, register B in 8085 is not variable, B means it is register B, content is variable.

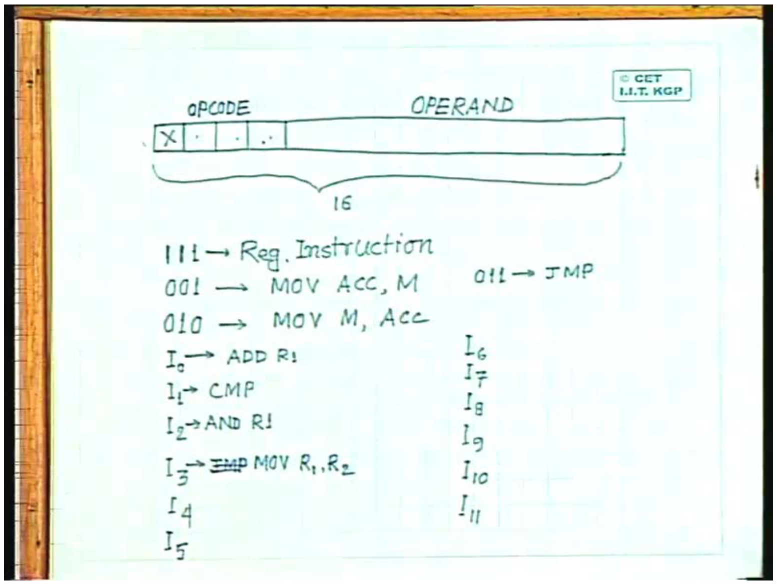
Similarly for this design, R_1 means register R_1. Its content can be varied but within the circuit, R_1 will have a unique identification. Similarly register R two will have a unique identification. So I have to have various data movement, instructions. So whatever is the sake of registers you have, I have to have movement instructions for all of them. That means I must have MOV R_1, R_2 so that I am able to transfer the data from register R_2 to register R_1. I have to have instruction MOV R_2, R_1. I should be able to transfer the data from register R_1 to R_2 as well. I have to have MOV instruction from say accumulator to accumulator from register R_1. I have to have instruction MOV to accumulator from register R_2 and the reverse. From accumulator also I should be able to move the data to these registers R_1 and R_2. So I have to have instruction MOV R_1, accumulator. I have to have instruction MOV R_2 accumulator. Yeah that is possible. I think these two instructions becomes clear (Refer Slide Time: 30:07). Is it MOV R_1 comma? Fine yeah, so let us remove these two. Sir then the data will be overwritten in case we are using the first two commands. I mean if we want to copy it from some accumulator to R_1 and we are cutting of these command. Yeah in that case the data will be overridden. It won't be saved? That won’t be saved. Yeah that is true but however this can be implemented. I mean these instructions can be implemented by these instructions. Yeah.

So next is these are the data transfer operations within CPU only. We will see that later on fine. So these are the data transfer operations within the CPU registers only (Conversation between prof and student: Refer Slide Time: 31:30 sir while removing these instructions R Mov R_1 from R_1 to… from accumulator to R_1 we want to load something… from accumulator to R_1 right…). So those are final adjustments that we can send it off, those are final adjustments that we can see later on.

So these are the data transfer operations which will be performed within the CPU only. Now in addition to this we must have data transfer operations between registers and the external one that is memory. So I can have instructions like this MOV, accumulator, memory that is I can read a data from memory to accumulator. I can also have or I should also have an instruction like MOV M, accumulator. When the data from the accumulator will be written into a memory location that this of the memory location will come from memory address register. Now for completion of our design, let us for the time being put these instructions as well. Optimization we will think of later on. So MOV R_1, ACC and MOV what was the other one? R_2, ACC. In addition to this, I can have additional instructions for example in and out instructions meant for I/O devices. However that is optional, if we design our system to be memory mapped I/O in that case these memory reference instructions are sufficient to do that job. So these are some of the instructions using which we can go for designing our CPU. MOV immediate instructions, you want to have that, fine let us put it.

So let us put this move immediate instruction, no problem. However that fine tuning can be done later on. Yeah as I said that for this designing, I am not considering all those things. It’s a very simple design, those are all fine tuning which can be done later on. This is just a hypothetical CPU that I am trying to design. Jump instruction we will load, jump address into program counter because you already know that whatever is the content of the program counter, the program execution starts from that address. So these are the instructions that we want to have in the CPU that we are going to design. Now once you decide what are the instructions, the next step is you have to decide that what will be the instruction format. So instruction format is you have to decide that how many bits or what will be length of every instruction. How many bits in that instruction will be allocated for allocated as output which will identify what instruction it is and how many bits will be reserved for operand. You see that in this case we have two types of operand. In some cases the operands are registers, for some of the instructions the operands can be memory locations.

(Refer Slide Time: 00:36:13 min)



So we will assume that let us have an instruction format something like this. How many instructions we have? 13 instructions, so we have here 13 instructions. For 13 instructions, we need 4 bits. So let us reserve 4 bits for instruction opcode. So these are the 4 bits which will decide which instruction it is. If I assume that the instruction length is a 16 bit, out of which these 4 bits have been used as instruction opcode. So the remaining 12 bits can be used as operand. Now there are various ways in which this operand can be used, operand field can be used. One way can be, though we have 13 different instructions but you find that these instructions can be grouped as registered reference instructions and memory reference instructions. So if I group the instructions into two, two such groups one is the group of register reference instructions, the other one is the group of memory reference instructions.

If I assume that one opcode will be used to identify say for example if this bit is a 1 1 1, I can say that whenever the opcode field is 1 1 1, the instruction is a register reference instruction and for register reference instruction I need not give any operand address because the operands are within the CPU registers themselves and there we have very very limited instructions, out of 13 I have two instructions which are actually memory reference instructions and the remaining 11 are register reference instructions.

Here I have 12 bits in the operand field. So what I can do is here when it is 1 1 1, I know that this is a register reference instruction and what instruction it is that will be dictated by the bit sequence within this operand field. That is one of the way. Here I have 11 register reference instructions, in the operand field I have 12 bits. So I can put it this way that if the LSB is one along with this bits as 1 1 1 then it is ADD R_1 instruction. If the next bit is one, all other bits in the operand field are zero or else in the opcode field, all are 1 1 1 then it is compliment operation. So that can be done because in this case this operand field does not have any other significance or otherwise because I am putting all these four bits here, so for each of these instructions I can have unique output that is also possible.

For the memory reference instructions like MOV M ACC, M or MOV M, ACC; I must specify that what is the memory address within the instruction itself. So for memory reference instructions I will assume that these 12 bits in the operand field will give me the memory address. So here I have this field as opcode and this field is an operand field. So in the operand field I have 12 bits. If I assume that the memory address has to be directly put into this operand field so that gives me a limit on memory addressing capacity of the CPU. That means now the address bus cannot be more than 12 bits. That is the maximum amount of main memory that can be accessed by the CPU is 2 to the power 12 that is 4 kilo bytes, 4 k locations. So we can assume that let me go to one case that my opcode field, when it is 1 1 1 let me take 3 of this 4. Let me for the time being forget about this. I will expand it further by making use of this bit. So I will use three bits as opcode field, so when 3 bits are 1 1 1 that is these bits 1 1 1 I, say that this is a register reference instruction. For any other combination of this opcode field, it will be a memory reference instruction but in this particular case we have only 2 memory reference instructions that this MOV ACC, M and MOV M, ACC. So I just need 2 opcodes. So I can say that let me put it as 0 0 1, when this will be MOV ACC, M and 0 1 0 which will give me MOV M, ACC.

Now when it is 1 1 1 then I have to identify the bits within this operand field which will uniquely identify this memory reference instructions, sorry registered reference instructions. So here I have bits in this operand field, when I call this entire thing as an instruction opcode, I have bits I_0, I have bits I_1, I have bits I_2, I have bits I_3, I have bits I_4, I have I_5, I have I_6, I_7, I_8, I_9, I_{10} and I_{11}.
These are the bits in the operand field. For memory reference instruction these bits will give you the operand address in the memory. For register reference instructions these will uniquely identify a register reference instruction. So I will say that when I_0 bit is 1, all other bits are 0. When I_0 bit is 1, all other bits are 0. This is equivalent to ADD R_1 instruction. So for ADD R_1, what will be the instruction opcode? These three bits will be 1 1 1 and in the operand field, it will be 0 0 0 0, all zeros LSB will be equal to 1.

Similarly when I_1 equal to 1, all other bits are 0 then this will be a compliment operation. When I_2 equal to 1 all other bits are 0, this will be AND R_1 operation. Similarly when I_3 is 1 it can be a sorry when it is a jump instruction then I have to give the jump address. So it has to be in this particular case, the address must be within the operand field. So this jump instruction cannot come here. So jump instruction has to be put within this group, so I will put it as this way say, 001 I have already used, 0 1 0 I have already used, let me put it as 0 1 1 in which case it will be jump instruction. Then with I_3 equal to 1, it can be MOV R_1, R_2 so like this you can fill up. So once you do this exercise, you will know that for every instruction I have an unique output. So once this opcodes are decided, now I have to go for instruction decoded design and finally the timing and control circuit design. So once these two are complete, your CPU design is mostly complete.

Next what you have to do is for each and every register, you have to set how the input output connections are to be made. As we said in case of a accumulator, since we have an instruction of compliment I have to have a provision of connecting Q bar output to D input but it should not be connected straight away. At the D input I must have some sort of multiplexer because the sources to the D input are different. That means there also I must have a selector which will select which of the inputs should be loaded into the accumulator. So those finer details of each and every data unit has in the CPU has to be done later. First you have to design that what will be the instruction decoder part and what will be the timing and control circuit part. So with this let us take some break then we will continue.