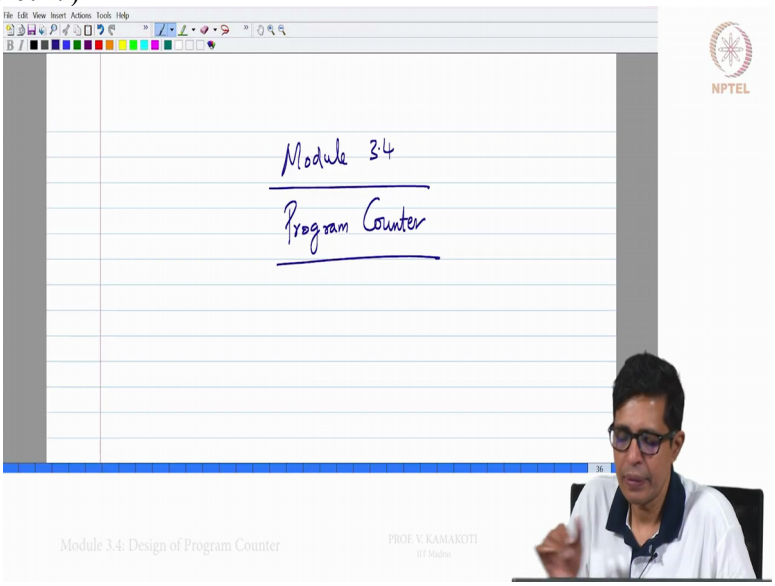
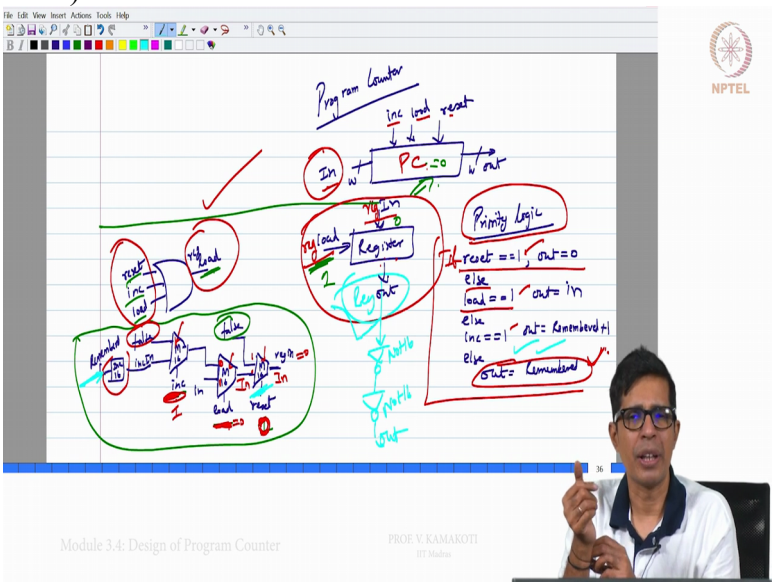
Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 3.4
Design of program counter.

(Refer Slide Time: 0:17)



So welcome to module 3.4 and in this module, we will be seeing one more memory element, the last that we will be seeing for construction of our architecture, namely the program counter.

(Refer Slide Time: 0:34)



What is this program counter? The use of this program counter, why we have got this name, all these things we will see at a later stage. But this program counter essentially works on this. So it is a register. This is basically a register tab so what will be the register that we have described in the previous module? That register basically has a load signal and an input and an output. When the load is 1 and you give a tick, whatever is there on the input will go to the register and when you give a tock whatever is there in the register, will come out as an output.

So this is how this is a clocked input clocked output register works. Right? So we will use this register so to basically construct this program counter. We will call this as the PC and what is this PC? Basically it is based on what you call as this priority logic. So the PC behaves like this. So there are 3 inputs to this PC, 3 control inputs to this PC, namely reset, load and increment. And there is a data input which is in which is W bits and out is nothing but what is stored in the PC. Now we will quickly tell you how these 3 signals behave. Right?

If reset equal to 1, irrespective of whatever is the value of Inc and load, so that is the important thing. If reset is equal to 1, independent of what is increment odd load, your what the content of your PC should become zero. That is what this says. Now we will see one by one. If reset equal to 1, your out is zero. If reset equal to 1, then I do not care what is inc, what is load? Your output will become zero. Right? So that is why I said, I give highest priority for reset. Right? If reset is zero, then only you go and see whether load equal to 1. When reset is zero and load equal to 1, then your out will become, your PC will store whatever is there on the In.

So when your reset is zero and load equal to 1 irrespective of what is Inc which is called increment your out will take the value, the value in the In will get stored in the PC. Now when both reset and load are 0 and the increment is 1, whatever is stored or remembered in this PC, that will get increment. So if I give increment equal to 1, whatever is stored in the PC will get incremented by 1. So if this PC stores a 10, then it becomes 11. And if all the 3 are zero, they said, load, increment, all are zero, then the output whatever is there in the PC will never change. Correct?

So this is a priority logic because reset gives the highest priority. If reset is zero, then load gets the next priority. If load is zero, then increment gets the next priority. If increment is also zero, then the value is remembered. You got this? So this is the basic way by which this program

counter works and there is a priority logic here. Now what is interesting now is how do we code this priority logic? How do we construct this program counter from the basic register? Of course, there is a register in which there is a in and out. Right? Let me call this as regIn and this as regload and of course out is out.

Now when will this register get loaded? Please note that the register will remain the same. The value of this register will remain the same when all the 3 fellows are zero. When your reset is zero, load is zero and the increment is zero, then what will happen? Your out will be the same. So your load, your regload need not change this load, need not change if all these 3 fellows are zero. Right? If one of them is one, then of course, your regload should change because your register can potentially get a new value. Right?

If reset is 1, more than the register will get a zero. If reset is load is 1, of course you will get whatever is given as an input. This input will come into this register and if you have increment this one whatever that register stored, that new value, that incremented value will come. So when I am constructing the program counter using register, I call the input to the register as regIn and the load input to the register as regload. That regload will become 1 if one of this increment load or reset becomes 1.

If all of them are zero, then the regload can be zero because whatever the value is there in the register need to be retained. It should not be changed. So the regload can become zero. So the regload is basically realised as an OR function of reset, Inc and load. OR of reset, Inc and load will give you regload. Right? So if reset, Inc and load, all are zeros, then this register will retain the remembered value. Let us go and see this priority logic. If all reset, load and Inc are zero, then out will remain remembered and this particular functionality is simply maintained by making this regload as zero so that in the next tick nothing will get changed in the register.

Now let us see when one of them is 1, then of course this register value is going to change. So your regload becomes one. Then what, how it is going to change? I need to find out how regIn behaves. This particular circuit which I am currently marking in green, will tell you how regIn will change, what will be regIn when the regload becomes. That is one of these reset, inc and load becomes 1 which makes regload as 1, then what would be regload? Now this is the priority, right? Your reset is, when the reset is 1, you get false. That means regIn is zero.

So when your reset is 1, your regload becomes 1. Because reset is 1, the r or this will become 1 and what will you get in regIn? False, zero. So in the next tick, your register will become zero. So essentially, your PC actually becomes zero. PC is used by, Vc is realised by using this regIn plus this circuitry. This whole thing that I am marking in a larger green square implements this PC that we have shown here. So now you see that when reset is 1, independent of load and inc as you see here, I am marking in again in red, independent of your load at inc, your regIn will become zero because that is the priority.

So reset gets the highest priority. That means reset is closest to the output in this diagram. Right? Reset is on the highest marks. Now when reset is zero, then you go in and your load is 1 okay, when your load is 1, then you get In here. And your reset is zero, of course this is 1, 0, reset is zero, so you will get In here. So reg will become In benefit is zero and load is 1. When reset and load, both are zero, then you consider this. If Inc is 1, then you have that remembered the value, right? The remembered value is given as increment 16 here. That will be increment plus 1. That will come here and then this will go.

So these are all mux 16. All these muxes you are putting here are mux 16 so that 16 bits will get routed. So when your reset is 0 and the load 0 and the increment is 1, then you basically get this value, right? So when your increment is 0, load equals, zero, reset is equal to 0, nothing happens. So we need not really bother about it. Okay? When your increment is 0, load is 0, reset is 0, register never changes. So I do not really care what is there, regIn here. So I can put a false [Inaudible 10:07] on there. So this is how the priority logic is coded. So what is driving regIn is this large circuitry which is a priority logic and the priority is basically established using different multiplexers.

The multiplexer closest to the output here, output of this circuit fear which is this M 16 which is this M 16 as you see here, is for reset and the next one is for load and next one is for Inc and this is how and this is basically how we are realising this. So you can basically call this out as Regout and this out will go through two inverters, two NOT 16s to basically get this output. And this Regout you can use it for, this Regout can be used for incrementation, the remembered value right? Because I cannot use out because an output variable cannot be used as an input.

So I will have a Regout and then Nout. We have seen in P2 why it is not possible. This is a small twist that we need to do here. But the way we have implemented this program counter, teachers use some lot of things in terms of how do you realise priority logic and what is priority logic, how do we realise it, in addition, we are also seeing this how the the different personalities of this program counter are basically realised using this. So this is how the program counter is made. So with this we sort of come to a conclusion on what are the different memory elements that we need to construct.

So we have seen in the previous module, we have assumed that D flip-flop, that is the smallest element and we have also told how D flip-flop can be constructed using transistors. Now we assume that D flip-flop exists. Now using the D flip-flop, we made bit. From bit, we made registers. From registers, we made RAM 8, RAM 64 and RAM 512, then RAM 4k, then RAM 16k. We can use, we can build much more RAMs like that. Then we also constructed the program counter and this all put together is enough for building our architecture.

So this also summarises what you need to do for your project 3A and 3B, there are 2 parts. In 3A, you will have bits, the program counter and the RAM8 and the RAM 64. And in the 3B, you will have RAM 512, RAM 4K and RAM 16 K. Each are all very very small code. Right? All those things we have explained. So the entire process will not take more than a couple of hours for you to finish this entire thing. So I suggest that sorry before you go back to module 3.5, you basically finish off this project 3A and 3B right? Thank you.