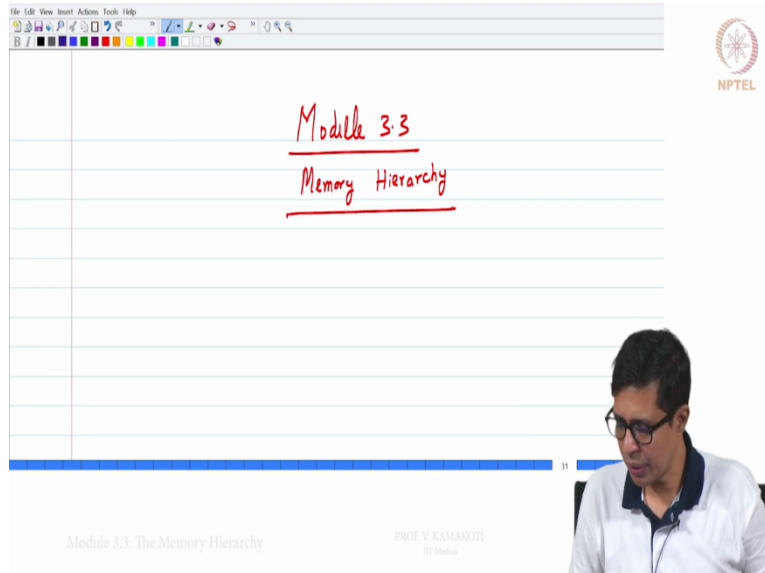


**Foundations To Computer Systems Design**  
**Professor V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Madras**  
**Module 3.3**  
**The Memory Hierarchy**

(Refer Slide Time: 0:17)



So welcome to module 3.3 in this way will talk about building the memory hierarchy. Memory is something which the system needs to remember over a period of time, so we have already seen what a flip-flop and latch is. Now we will basically now go back and start looking at how do we use this flip-flop and latch to build components in the circuits that will remember the values over a period of time.

The latch and the flip-flop basically remember the latch actually remembered for half a clock cycle whereas the flip-flop actually remembered over 1 full clock cycle, so whatever came as an input at this Pasedge of the clock that was remembered till the next Pasedge. Now with this feature can we go and build circuits that can remember things for over a period of time. So that is what we will be covering in this memory hierarchy.

(Refer Slide Time: 1:28)

Module 3.3: The Memory Hierarchy  
PROF. V. KAMAROTTI  
IIT Madras

Now let us look at the first thing. As I told in the previous module the D flip-flop is assumed to be available as a basic building block. Now what we have constructed here is called BITS, this is called a bit to be more precise this can store one bit. What is a functionality of this? When the load is one, the load input is one at the positive edge of the clock that is Tick, tick of the clock.

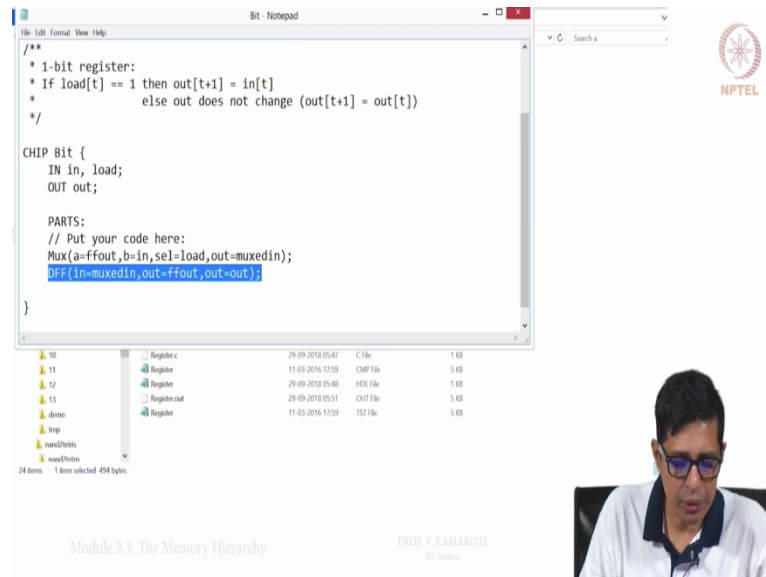
If the load is one, so at the positive edge of clock if the load is one then IN value moves into DFF and appears at out at tock of the clock. So the clock was as tick tock tick tock tick tock, so at the tick of the clock which we assume is a Pasedge if the load is one whatever is given at that input at the tick of the clock that will move into the D flip-flop and appear at the out at the tock of the clock, okay.

If at the Pasedge of the clock if the load is actually 0 then whatever was inside the flip-flop that will get again restored, so the flip-flop basically remembers its previous value. So the value stored in this will change only when the load is one and the clock becomes tick and that changed value will be reflected at the out at tock of the clock. So the clock goes as tick tock.

If the load is 0 then the flip-flop remembers across all tick tock till the load becomes one it is not just enough when the load becomes one, the clock should also tick for that value in the IN to go into the bit, right? So this is the basic functionality of this what we call as BIT storage. This BIT storage will be used to construct larger memory elements as you see here.

Now before proceeding further with larger elements, let me show a very small demo of this bit. So bit is project 3A you will see that and this is the code, so this code we need to basically understand, so I will just open the bit code for you.

(Refer Slide Time: 5:11)



```
File Edit Format View Help
Bit - Notepad
/**
 * 1-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 * else out does not change (out[t+1] = out[t])
 */

CHIP Bit {
  IN in, load;
  OUT out;

  PARTS:
  // Put your code here:
  Mux(a=ffout,b=in,sel=load,out=muxedin);
  DFF(in=muxedin,out=ffout,out=out);
}
```

Module 3.3: The Memory Hierarchy

PROF. V. KAMAKOTTI  
17:00:00

Yes, so I assume the D flip-flop and now very carefully you have to know this is something which is very important the output actually has to come and feedback into the input and if you had seen in module P2 the output cannot be used as an input. This is and out is actually and output variable of your chip and that cannot be used as an input. To facilitate this that D flip-flop is constructed in such a way.

(Refer Slide Time: 5:48)

```
Bit - Notepad
File Edit Format View Help
/**
 * 1-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 * else out does not change (out[t+1] = out[t])
 */

CHIP Bit {
  IN in, load;
  OUT out;

  PARTS:
  // Put your code here:
  Mux(a=ffout,b=in,sel=load,out=muxedin);
  DFF(in=muxedin,out=ffout,out=out);
}

// Note: The code in the image has some typos: 'muxedin' and 'ffout' instead of 'muxed_in' and 'ff_out'.
```

Handwritten notes:

- Load = 00
- Remembers its previous value
- scope clk if the 1, then 2nd value moves into DFF and appears at out at "back"

Module 3.3: The Memory Hierarchy  
PROF. V. KAMAROTTI  
IT Madras

The D flip-flop is constructed in such a way that there are, as you see here there are 2 outputs out equal to out is the final output but there is another out which is called is FF out, right? So you could have 2 OUTS you can define one is the FF out, so this out is the final output while this out can be tapped to take inside.

(Refer Slide Time: 6:12)

```
Bit - Notepad
File Edit Format View Help
/**
 * 1-bit register:
 * If load[t] == 1 then out[t+1] = in[t]
 * else out does not change
 */

CHIP Bit {
  IN in, load;
  OUT out;

  PARTS:
  // Put your code here:
  Mux(a=ffout,b=in,sel=load,out=muxedin);
  DFF(in=muxedin,out=ffout,out=out);
}

// Note: The code in the image has some typos: 'muxedin' and 'ffout' instead of 'muxed_in' and 'ff_out'.
```

Diagram of a D flip-flop:

- IN is connected to the D input of the DFF.
- load is connected to the clock input of the DFF.
- The output of the DFF is labeled 'out'.
- The output of the DFF is also connected to a multiplexer (MUX) input.
- The MUX output is labeled 'muxedin'.
- The MUX output is also connected to the D input of the DFF.
- The MUX output is also connected to the 'ffout' output.

Module 3.3: The Memory Hierarchy  
PROF. V. KAMAROTTI  
IT Madras

So just very quickly we can see here, this is what we are seeing, now we can see that D flip-flop actually takes the Muxed in this is the wire muxed in as you see here and the out is out, this is this line this wire is actually called as FF out flip-flop out this FF out is fed into a multiplexer that multiplexer takes this is FF out as one of the input and in as another input in

as another input and the select is the load and the out is mixed in. So this will basically function in the way we have showed this, right? So this is the basic definition of a bit.

(Refer Slide Time: 7:16)

The screenshot displays a digital logic simulator interface. On the left, there are panels for 'Chip No.', 'Registers', and 'Messages'. The main area shows a circuit diagram of a DFF (Data Flip-Flop) with handwritten annotations in red and blue. The input is labeled 'load' and 'data in', the output is 'out', and there are labels for 'clock' and 'set/reset'. A 'Windows Journal' window is overlaid on the right, showing a handwritten diagram of a DFF circuit with similar labels. The NPTEL logo is visible in the top right corner. At the bottom, there is a video feed of a man with glasses speaking, and text identifying the module as 'Module 3.3: The Memory Hierarchy' and the speaker as 'PROF. V. KAMAROTTI, IIT Madras'.

Now we will run this bit to explain we will show you the demo here, so the bit is load at this file bit, right?

(Refer Slide Time: 7:29)

Module 3.3: The Memory Hierarchy  
PROF. V. KAMAROTTI  
IIT Madras.

And now we will take the script for bit and loaded the script and it is here, so what it will do? Load bit (0) (23:10) load bit that out without compared without list, let us go step-by-step, let us run this step-by-step. So we are sitting into 0 load to 0 tick that means since load is 0 whatever is inside should be remembered. Let us assume that before starting of the simulation the value inside the flip-flop is 0.

Now when I say set In 0 load 0 nothing should happen the old value should remain the same. So let us start this simulation step by step, let us keep watching what is happening here okay. Let us do this step-by-step so initially in a 0 load is 0, I do a tick output tock output nothing. Now IN is 0 load is one and I have done a tick now I'm doing a tock nothing should change because out is already 0 IN should be equal to out though load is one, so this is what...

Now IN is one and the load is 0, I am doing a tick nothing will happen I'm doing a tock nothing will happen, so till that out is remain to be 0. Now let us see this I am making IN one then I'm going to make load one the moment I tick what will happen? Your flip-flop out should become 1 and when I tock the output should become 1 till then everything has been 0.

(Refer Slide Time: 9:46)

Module 3.3: The Memory Hierarchy

PROF. V. KAMAROTI  
IT Madras

Now I'm setting IN as 1 load as 1 my muxed in since I gave tick my muxed in, so IN is one here load is one here, so the muxed in has become 1 on this tick.

Now what we need to do on the tock the output should also become 1, now I do a tock now as you see the output has actually become 1. Now I'm making IN is 0 load is 0 and I do a tick now as you see the output has actually become 0. Now I'm making IN is 0 load is 0 and I do a tick now as you see the output has actually become 0. Now I'm making IN is 0 load is 0 and I do a tick now as you see the output has actually become 0. So IN is 0 load is 0 note that though my IN became 0 my muxed input still remains one because your load is 0 here and then I now go and do a tock still the output will not change.

Now I making IN 1 and load 0 again nothing should change, so the output remains one. Now I do IN 0 and load one and so my muxed input should become 0 now because load is one at the end of this tick x muxed input became 0 and now at the tock your output should also become 0, yes.

(Refer Slide Time 11:04)

Module 3.3: The Memory Hierarchy

PROF. V. KAMAROTTI  
IT Madras

So what you do is that these inputs that are given here are called clock inputs. So in a chip where there are clocked inputs and clock outputs. The clock you set a input and do a tick of the clock, it is called tick as you see here then the input will enter the chip and then the output will be computed but the output will be revealed out when you do a tock, so this is how the clock chip works.

Tick it enters tock it basically comes out as a output, so the entire script that you all seeing here which is already written for your (()) (11:40) it just sets the input, inputs are IN and load it does tick and checks what is the output is then it does tock then tick which says the input, it is tick then tock the output will come, so set the input do tick then tock then again set the input do tick then tock, so the outputs will be coming out again and again.



(Refer Slide Time: 13:12)

Module 3.3: The Memory Hierarchy

PROF. V. KAMAROTTI  
IIT Madras

So this is how you can simulate these clock sequential circuits between a tick and the tock the value is remembered and when the load is 0 the value is remembered for lot amount of time. If the load is one the circuit will take in the new fresh input on a tick and tock but when the load is 0 the bit will retain its value for long amount of tick and tock as long as load remain 0 the value will be remembered irrespective of the number of ticks and tocks. So this is how we construct one element one bit that could be remembered for as much ever time we want, right?

By adjusting the value of load I can make it remember something I can make it load a new value, right? So this is the basic element and we have shown the simulation of how this element works, right?

(Refer Slide Time: 13:13)

W-bit  
Load=1 The values in the "in" should get stored on tick, else old value should be remembered

Register

Load

Bit Bit Bit ... Bit

in(0) ... in(W-1)

out(0) out(1) out(2) ... out(W-1)

Module 3.3: The Memory Hierarchy

PROF. V. KAMAROTTI  
IIT Madras

Now the next thing that we will see is the Register. So how do you make a Register? So I have to make W bit register normally we will be using the register here. What is a register? Again the same thing W bit register is works as follows, if you make load equal to 1 then the value in the IN should get stored else old value should be remembered load is one and again the value in the IN should get stored on tick else old value should be remembered.

So what we did for one bit we need to now do for W bits, so W bits make, so one bit was constructed using a D flip flop, now we take that bit and repeat it W times to basically get this W bit register. So that is very simple so instantiate W the bits, say W times inside the same load will go to all the bits, so bit has 2 inputs, right? The load and the IN and we want to store W different bits using this register, so the load is common for all the bits instantiation but the input will be in 0 (0) to W minus 1 for each of this bit.

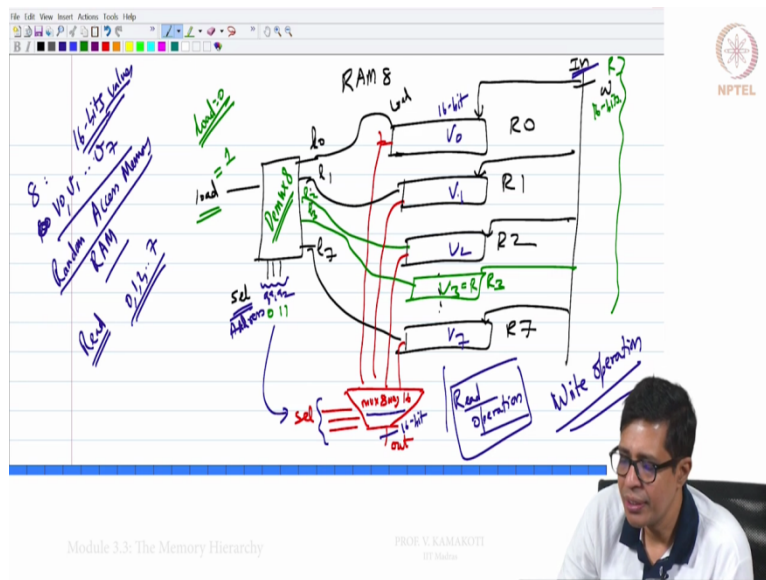
And the output you will be tapping from each of the bit that will be out 0 to out W minus 1. So this is how I construct a register where in W bits can be stored when the load is one and the W bits can be retrieved, okay. They are being stored here and then we can take the value of this W bits at any point of time. So this is how we construct a register.

A register is the 1<sup>st</sup> in this storage hierarchy inside a processor the register remembers some value over a period of time and this can remember say W bit value in the real project we will be doing a 16-bit register, so it can basically remember 16 bits across many ticks and tocks, so if your load becomes one and you give a tick whatever is there on the input of the register

will go and get saved and if as long as maintained load equal to 0 that value will be remembered irrespective of number of ticks and tocks you're going to get of the clock.

So register is a clock sequential storage element which can store multiple bits and use as the bit primitive that we have just coded, okay.

(Refer Slide Time: 16:25)



So now the next thing that we will do is that we have now designed the register, now how do we design a memory? Memory will store collection of values, so a register is storing one 16 bits, a memory can store multiple such 16 bits, so what is a memory here? Suppose I want to store say 8 16 bits values. Let me call those values as value 0, value 1 to value 7. Let these values be stored at R0, R1 to R7.

So V0 will be stored in R0, V1 will be stored in R1, V2 will be stored in R2, V7 will be stored in R7. Now I want to do 2 operations on this, so I'm trying to remember a collection of values that is why we call it as a random access memory or RAM. So V0 is stored in R0, V1 is stored in R1, V2 is stored in R7 each is a 16-bit number as you see here, right? So with this memory I want to do 2 operations.

The 1<sup>st</sup> operation let us call it as a read operation I want to give that address, right? I want to give that address and I basically want to read what is there in that address. So how many addresses I have? I have 8 different addresses namely address 0, 1, 2 till 7. In binary to represent that address I need 3 bits these are this select address bits that you see here, the 3 bits I am marking in blue A0, A1, A2 the same 3 bits are given to the select.

Now whatever I Mark in red here is responsible for the Read operations, right? So how do you do the read? So V0, V1, V2, V7 they are all 16-bit values and 8 such value you just put a muxed 8 way 16 and give the address if it's here, you will get an output which is again 16 bits and that output will correspond to whichever value you want to read. So if I put 000 as this address bit here then I will get the value of V0.

If I put 010 here I will get the value of V2, if I put 111 I will get the value of V7, so the read operation is basically taken care of by this circuit which is shown in red which is a mux 8 way 16 circuit and you give the address you can value from this. So this is basically called the read operation. Similarly on the other side I will have something called the write operation.

So what we do in the write operation here? Write operation is very straightforward now the black circuit that I am putting here will be responsible for the write operation. Whatever you want to write you put on IN this as I am showing in green here, whatever you want to write you put on the IN this is again W bits or 16 bits for as this case. So that IN will be available for register R0, R1, R2 to R7, so the IN available to each one of these registers. So then you use that D mux here and give this load signal to this, the load will become 1.

If I want to do a read the load is 0 if I want to do a write the load becomes one, so suppose I want to write that value of IN to say V3, right? Let me put V3 here which is R3 this is L2 L3, so I give 011 as an input w here what will the D mux do it will make L3 1 and L0, L1, L2 to L7 remaining things are 0. So exactly L3 alone will become 1 that means the load of this particular register R3 will become 1 and then when I gave a tick whatever is there on the IN will just get stored in the V3.

So suppose I want to store some this V3 will now become, suppose I put R some value R, it's a 16-bit value now this V3 will become R, so I basically put a D mux to which I feed a load signal and whichever address I want to write I make load is equal to 1 give that address to the D mux then the corresponding load line will become 1 for that address alone the remaining addresses the load line will become 0 and now whatever I want to write value that I want to write I'm giving in the in wire as you see here.

And that IN will be fed to all the register and so what happens is that register which I have selected using this D mux by that load line that register will basically and you give a tick then that register will get the value R. So a D mux is basically used for the write operation and a

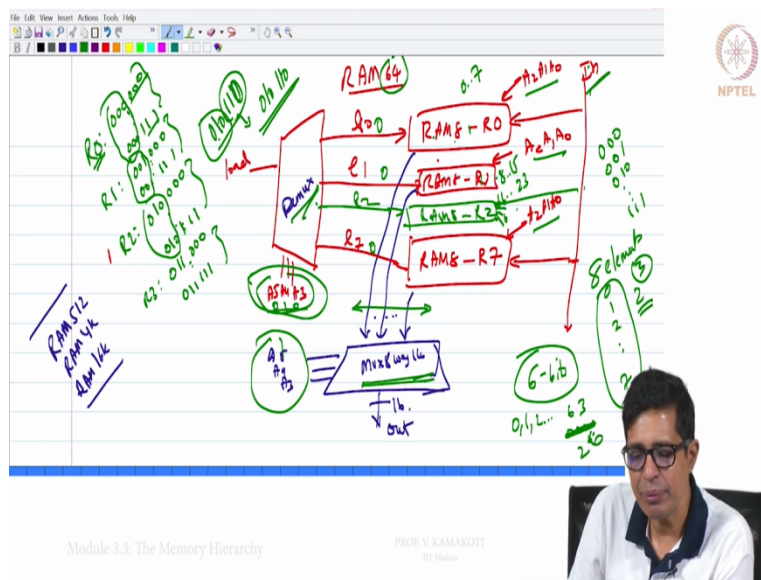
mux is actually use for a read operation this is how using 8 registers we can basically construct a RAM8.

8 8 16 bits it can store this is a random access memory and more importantly what you do with the random access memory you give a address and you go and write into it are given address and you go and read from it if you give an address and you want to write into it we also make load equal to 1, so you give the address, you give the data in the IN and you make load equal to and give a tick then in that address that whatever data you are given in IN will get stored.

Similarly if you give just the address and load of course is 0 and you gave a tick and then a tock then whatever is stored in that address will get retrieved as a note? The multiplexer is responsible for retrieving the value that D multiplexers are responsible for you know writing into it. So in this case the D mux 8 can be used and here mux 8 16 can be used.

So this is how we built a RAM 8 from registers. So we have built a register using bits now we have constructed a RAM 8 using these registers.

(Refer Slide Time: 25:47)



Now the next that thing we will be doing is, we can construct a RAM 64 using RAM 8 and that is very easy, right? So now we construct many RAM 8s here, RAM8 R0, RAM8 R1 till RAM8 R7 so instantiate 7 RAM 8S here so if I want to address 64 elements, right? If I want to address 64 elements if I want to address 8 elements I need 3 bits, if I want to address 64 elements I need 6 bits, 8 elements the address will be 0, 1, 2 till 7, so 8 different commendations which is 2 power 3.

So I need 3 bits, 3 bits would be 000, 001, 010 till 111, so if I want 64 then it becomes 6 bits, it is 0, 1, 2 to 63 which will be 0000 this is 2 power 6, 64 is 2 power 6. 64 is 2 power 6, so I need 6 bits, right? So how do you build a RAM 64, so you put 8 RAM 8S with essentially becomes 8 into 8 is 64. Now there are 6 the address bits, so there is a6 A5, A4, a 3, a 2, a 1, a 0 this a5, A4 and A3 will tell me which RAM 8 I need to select.

Note that in the first RAM 8 the address 0 to 7 will be there, in the 2<sup>nd</sup> RAM 8, 8 15 will be there in the 3<sup>rd</sup> RAM 8 it will be 16 to 23, right? So let us now look at it very carefully in RAM 0 the address is 000000 to 000111 will be there in r1 in RAM 8R1 you will have 001, 000 to 001111 will be there. The RAM 2 will have 010 000 to 010111 in RAM 3 it will be 011000 TO 011 111, so these are all the address is that will be there.

So the first 3 bits identifies which RAM I need to go which RAM 8 I need to go should I go to R0 or R1 or R2 the first 3 bits will tell me, first 3 bits means A5, A4, A3 the most significant bits. The last 3 bits namely A2, A1 and A0 will tell me within that RAM which location I need to access, right? So within that RAM which location I need to access. So how do you construct RAM 64 now?

You put a D mux with A5, A4, A3 that will give me L0 to L7 which you select which RAM I need to take, right? And for each of this RAM modules put the A2, A1, A0, A2, A1, A0 for all of them and within that RAM it will select which location, right? So you instantiate a times this RAM 8 and for each of this RAM 8 there is a load that we need to give that load will basically come from the D mux which is selected using A5, A4, A3, so that will tell me which RAM should I select as we have seen here.

And within each RAM I use A2, A1, A0 to select the address within the one of those 8 within those 8 that are store there and so each RAM will give me some value, right? And then again I use this A5, A4, A3 to select from whose RAM value I need to take. So the read operation works like this. A5, A4, A3 will tell you from which RAM I want the data and A2, A1, A0 will tell me from that RAM which one of those 8 data I need.

So I give A2, A1, A0 to each of these RAMs, so each one will give me the corresponding A, so suppose I want to read 010 111 or 110 this 110 will be given to each one of these 8 RAMs, so the 6<sup>th</sup> location 110 is 6<sup>th</sup>, the 6<sup>th</sup> location for each of this RAM will come here. So I will get 8 such inputs from those 8 I need to select the 2<sup>nd</sup> one that is 010, so this mux 8 way 16

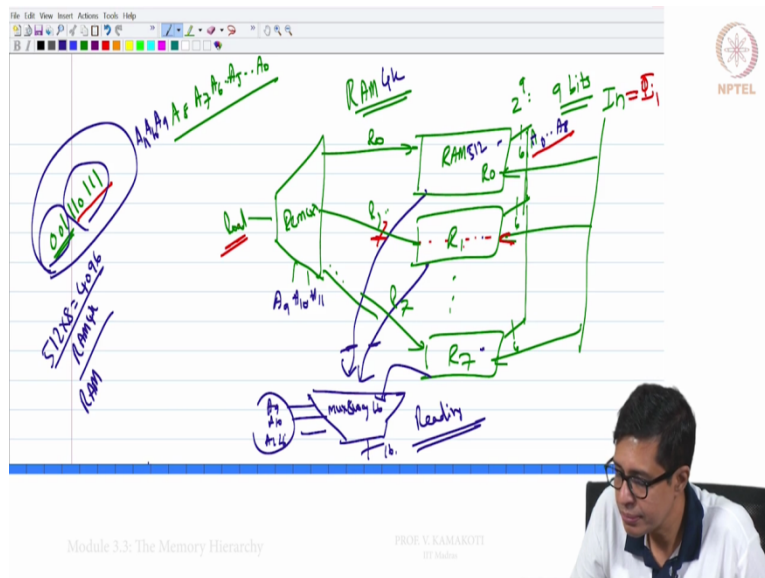
will select me the 2<sup>nd</sup> of those values the 6<sup>th</sup> location value coming from the 2<sup>nd</sup> RAM will be selected by this marks 8 way 16.

So this blue color what I have put here is going to be responsible for reading from this RAM. Similarly for writing into that RAM the same way I put this input here this In as I am marking in green here is connected to all these registers the A5, A4, A3 in the D mux that is there that will select which RAM should I write. So if I want to write into 010 110, so 010 is given here. So that 2<sup>nd</sup> RAM L2 will basically be enabled all the other things are disabled.

Now this IN value is fit to all of them, so I want to write into the 2<sup>nd</sup> RAM 6<sup>th</sup> location, right? Sorry 2<sup>nd</sup> RAM 6 location, correct. And so here A2, A1, A0 is given as 6, right? 110 is given into the 2<sup>nd</sup> RAM, so the load is given there, so in the next tick the 6<sup>th</sup> location of the 2<sup>nd</sup> RAM will be written the value that is given in IN, right? So this is how the right operation happens into the RAM 64, right?

So what we have done now? We have basically built a RAM bit, from D flip flop we got it bit from the bit we got a register and we used 8 such registers to build RAM 8, now use that 8 RAM8s to build RAM 64, right?

(Refer Slide Time: 32:47)



Right, now we will use 8 such RAM 64 to build RAM 512, so we want to build RAM 512. 512 is  $2^9$ , so we need 9 bits to do that. Now we will put, we have already done 64 so we put RAM 64 we will call it as  $R_0, R_1$  to  $R_7$ , to address this we need 6 bits, 6 address bits, so we have 9 bits, right?  $A_8, A_7, A_6, A_5$  to  $A_0$ , right? So we will give  $A_5, A_4$  to  $A_0$  for each of them.

Each of them will get that  $A_5, A_4$  to  $A_0$  and there will be an  $In$  which is coming to each so here again I put D mux where in I put a load signal and I put  $A_8, A_7, A_6$  select signal and this will go for  $L_0, L_1, L_7$ , so suppose I want to write into location 001 110 111, right? So 001 will make  $L_1$ ... So 001 is fed into this D mux, so this will make  $L_1$  to and in  $L_1$  this  $A_5$  to  $A_0$ , so this 110 111 that is the location given here.

In that location whatever you're putting on  $In$  let me say  $I_1$  in that location within  $R_1$  this  $I_1$  basically gets note. Note that  $R_1$  is enabled,  $R_1$ 's load is enabled and whatever value  $In$  will get it, right? So this is for the right operation where you make load equal to 1 then it will whatever is there in  $In$  will go and sit into that particular location, okay. So internally RAM 64 will take 6 bits and it will store in the location we have already seen how RAM 64 works.

Now again for read operation each one of them will give me 8 16 16-bits again I make so mux 8 way 16 and I give this  $A_6, A_7, A_8$  and I get this 16-bit. So suppose I want to read 001 110 111. 110 111 is given to  $R_0, R_1$  to  $R_7$  they will give the corresponding 8 bits in that location out of all the 110 111 locations that 8 locations I am saying I need the one from the RAM number one.

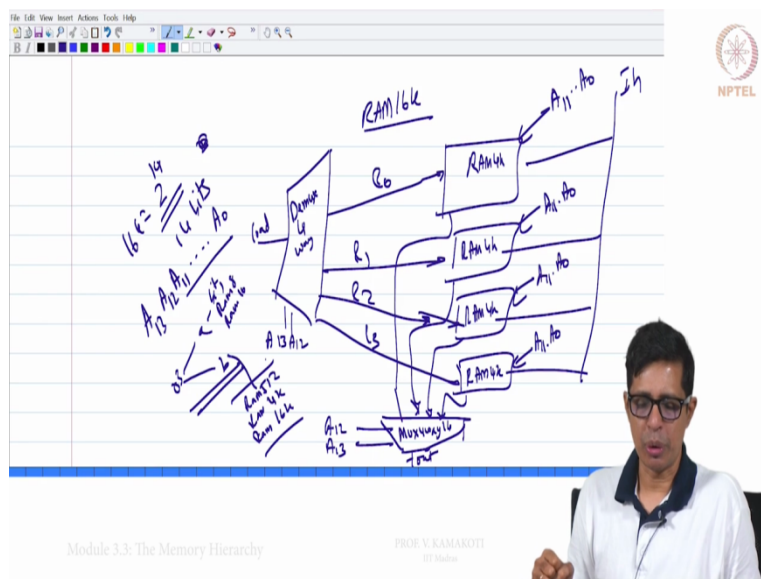


So this 001 will select the output of the RAM number one and I will give it to you, right? So this blue fellow will be responsible for reading. So using RAM 64 we have built RAM 512.

Now using RAM 512 into 8 is 4096, so I can build what you call as RAM 4K, how do I build RAM 4K? RAM 4K actually needs 12 bits, so we need A9, A10, A11, so the same structure instead of RAM 64 you make it RAM 512. This is a RAM 4K, right? You make it RAM 512 here and here you give from instead of A0 TO A5 you give from A8, A0 to A8 9 bits and here you give A9, A10, A11 A9, A10, A11.

Same circuit just instead of 4K we make it 512 and instead of last time we had RAM 64, instead of 64 you make it 512 each one of them gave 9 bit address rather than 6 bit address and this select this is the 3 inputs, so this will give me RAM 4K it will exactly work how the RAM explanation for RAM 512 how did we build 512 from 64 the same explanation now valid for 4K for 512.

(Refer Slide Time: 38:30)



Now using RAM 4K we can build RAM 16 K, so how do you build RAM 16K? We just need 4 of this 4K, so RAM 16 K can be built as follows. 16 K is 2 power 14, right? 2 power 10 is 1K, 2 power 4 is 16K, 16K is equal to 2 power 14, so we need 14 bits address bits starting from A13, A12, A11 to A0, okay. Right? So you put RAM 4K here RAM 4K4 times, so that will give me RAM 16 K for sure and give A11 to A0 12 bit address for each.

And whatever output is coming here this is going to be mux 4 way 16 and you are going to get an output here 16bit output and you give this A12 and A13 as inputs and then here you have D mux, so you basically give a load signal here this is D mux 4 way and you give your

A12 and A13 here and this will go for L0, L1, L3 and of course the In since 16-bit is connected to all of them.

So set the address bits made load equal to 1 then the corresponding thing and set your input data give a tick your data gets entered and similarly gave the address and give your load make load 0 give the address and give a tock then your output basically gets this. So this is how read/write operations can happen on a RAM 16K. So what we have done in this module is we started with a bit, we then made a register then we went again and then made RAM 8 which can store 8 16-bit values.

Then made RAM 64 then 512 then 4K then now 16 K, so all this. So this is there as project 03 a directory, project 03 b directory you can do this different RAM so project a basically as bit and RAM 8 and RAM 16 this b as RAM 512 4K and 16K, right? And there are just routines for this you can just go ahead and do this. So with this we have talked about the building the memory hierarchy.

We will now go to the next module where we will design something called a program counter.