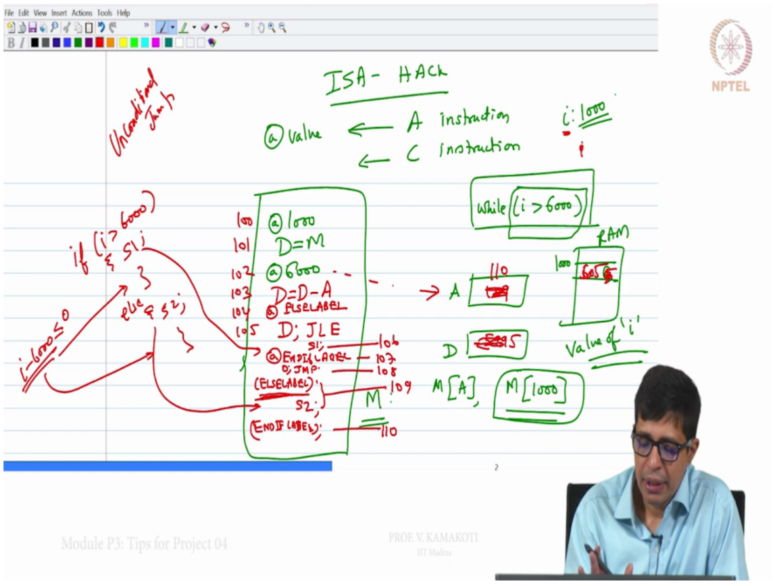
Foundations to Computer Systems Design.
Professor V. Kamakoti.
Department of Computer Science and Engineering.
Indian Institute of Technology, Madras.
Module P3.
Tips for Project P04.

Welcome to module P3, this is again a facts on project. As of now you should have completed P01, P02, P03A and P03B and your value should be working, all your tests should have passed. If you have any issue in that, please do put it on the discussion forum, do not give up, you can make it work and it is very simple. Now what we will do now, is to look at P04. Now working with P04 is extremely important for you to have understood this entire module 3.1 to 3.6. So we have there quite a bit of work there, especially the 3.5, 3.4, 3.5 and 3.6, where we started talking about the instruction set architecture.

Now this exercise which you are going to do, it is very very simple, it hardly takes another 1 hour for you to complete 2 exercises that are part of P04 and that will give you a complete understanding of the instruction set architecture that we have been talking of so far. And it will also give very good insight into assembly programming. Assembly programming is a must for you to understand so that later when you do embedded systems or any sort of computing, when you want actually performance, there may be a need for you to go and start looking at the assembly program.

So irrespective of the discipline that you are studying, whether you are computer scientist or electronics, anybody who wants to walk into the area of great system for information security, there is a need for us to have an understanding of assembly programs work, how assembly languages are formed. So this particular project will give you tremendous insight into those issues. And so please take this project very seriously and let us do that. So this will also sum up, many of the things that you may not have followed in your model 3.4, 3.5 and 3.6, right.

(Refer Slide Time: 2:29)



Now let us look at, so as far as the instruction set architecture of our hack goes, there are 2 types of instructions, what is the at A or whatever, and value, this is called the A instruction and there is another instruction which is called the C instruction. Already I told you, right, suppose I want to do, while i is greater than 0 or while i is greater than 6000. Suppose I want to do while i is greater than 6000. That means I basically want to 1st check this condition i is greater than 6000. So I can basically say, let me say that i, for our understanding is that location 1000, memory location 1000, every variable will be associated with a memory location.

So, I will just say at 1000, when we say at 1000, there are 2 registers, namely A and B, the A register will get 1000. So, this assembly, I am writing their family program. The moment I say M in the assembly program, it refers to M of A, that is M of 1000. For the moment I say M here, it refers to the content of the memory location 1000 which is nothing but the value of the variable. So when I have variable, that is, there are 2 attributes like, if you are a human being, you have, for example you have name and age as attributes. Like that the variables have attributes.

So if I have variable i for example here, then it has an address and it also has a value. So i, this 1000 is the address of i, the content of this 1000 is the value of i. So there is main memory her and there is 1000 here, right. Now let that content be 5 for example, let us start here. So if, the moment I say M, this is M of the address that is totally A, M of A, so M of 1000, so it will be 5. So M, essentially if I use the value M here, it is, it essentially uses a value 5.

So if I say M equal to M +1 here, what essentially happens is your 5 now becomes 6. M equal to M +1 means your 5 actually becomes 6. If I say D equal to M, essentially that meets the register D gets the value 5. Now I can say at (())(5colon31) 6000. The moment I say at 6000, the A register basically gets the value 6000, right. Now will now say D equal to D - A, D equal to D - A, what is D equal to D - A? Now, your 6000 is equal to, sorry 5 equal to, now you are new value of D will now become 5-6000, essentially this will become -5995.

Because D was originally 5, A was 6000, so 5-6000 will become this, right. Now, now we can go and check it, now we will say that suppose this i is, so let us say if i is greater than 6000, do something, else do something else. So, that means say this is statement one set of statements, this is statement 2. Now, here if this is i is going to be less than A, i is going to be less than 6000. I have to go and do some statement S2. So we will do some label here, that, the else label, right and hear all the statements S2 will be there, here all the statements S1 will be there, right.

So when I say D equal to D - A, now we will go and say at else label, else label is a symbol that we have written and we will put S2 here. Now, I go and check D semicolon if it is, so if it is J is less than or equal to 0, J L E, then I put this. The Machine language corresponding to S1 here, the machine code corresponding to S2 here. Now I will say end-if label, okay. So at this point after you finish S1, you can put at end-if label and you said 0 semicolon jump. Okay, so what happens here, 1st I say at 1000 D equal to M, so 1000 i is mapped, so D now gets the value of 5, which was to start with it was 5, okay.

Now add 6000, so A gets the value 6000, when I say D equal to D - A, A gets the value -5995. Now if the value D - D - A is negative, that means I have to execute else because I am comparing i with 6000, so I will subtract that 6000 from i, if it is less than or equal to 0, then I have to execute the else part, if it is greater than 0, then I have to execute the if part. So I just put at else label, which is, which is the address of this location here and I just checked D JLE.

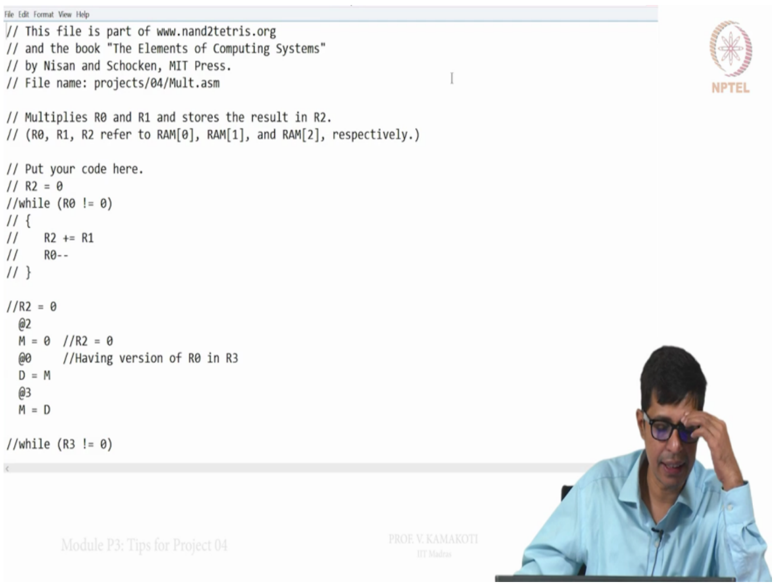
If D is less than or equal to 0, JLE means less than or equal to 0, then what should I do, I should jump to the address in A. So when I say at else label, so let us assume that we have started with 100 here, this will be 101, 102, 103, this is 104, 105 and then this S1 let us assume is a single statement 106, 107, 108 and so this S2 again let us, so else label is 109 and end-if label is 110 so, the moment I say at else label, your A now will become 109. So, now I say D JLE, D is negative, so it is less than or equal to 0.

So essentially JLE, tough in leather or equal to wear, the ad is given in A, that is 109. So I will come to else label, execute S2 and go off. But suppose this is possible, suppose i was say instead of 5005 or 6005, right, so when I subtract, this would have given 5, +5. So your D will not be negative your, then what you do, then you execute the statement S1, then you say at end-if label. The moment I say at end-if label, end-if label is 110, so 110 will be loaded here and 0 colon, jump, 0 semicolon jump essentially says no condition to be checked, just jump, this is called an unconditional jump, there is no condition to be checked here.

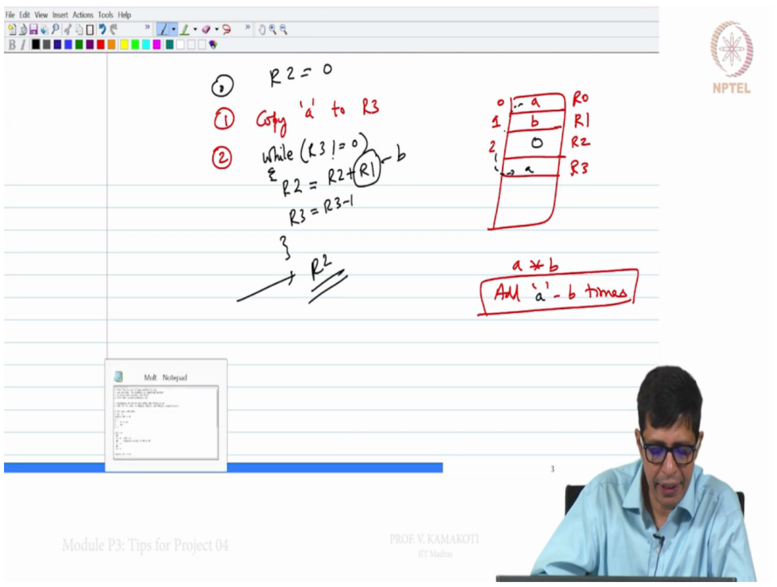
So your code actually comes to end-if label and finishes. So this is how we can execute if we can translate and if-else statement, right. So the things that we need to understand here are how do we handle the, there are some, how do we handle jumps and how do we handle conditions. So handling jump is the major thing you, we have seen both conditional jumps and unconditional jumps. So this is the way by which the ISA hackers use for translating your C program into this ISA language, okay.

Now, as a part of your project, so this gives you some basic understanding, as a part of your project, we have to do 2 programs, so we will explain one program now and the other program the structure we will just discuss here so that we take it forward from that. So the program is something like this, we will just go to this. So this is the 4th project here, there is, so we just take this mult program there is multi dot ASM that you will see there, you can open that and that is where you will see this is the code.

(Refer Slide Time: 12:58)



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/04/Mult.asm

// Multiplies R0 and R1 and stores the result in R2.
// (R0, R1, R2 refer to RAM[0], RAM[1], and RAM[2], respectively.)

// Put your code here.
// R2 = 0
//while (R0 != 0)
// {
//    R2 += R1
//    R0--
// }

//R2 = 0
  @2
  M = 0  //R2 = 0
  @0     //Having version of R0 in R3
  D = M
  @3
  M = D

//while (R3 != 0)
```



So I just put this code here. So, it basically says that I have R0 and R 1. So in the RAM I have R0, R1 in the location 0 and 1 and that is R2. Let R0 Store A, R1 store B, write a program that will take A and B and computer A into B and so writ in R2. So the way we are going to do is, please note that our ALU cannot do multiplication, the ALU that we have designed can do only addition, so it can do X + Y or X and Y, it cannot do multiplication.

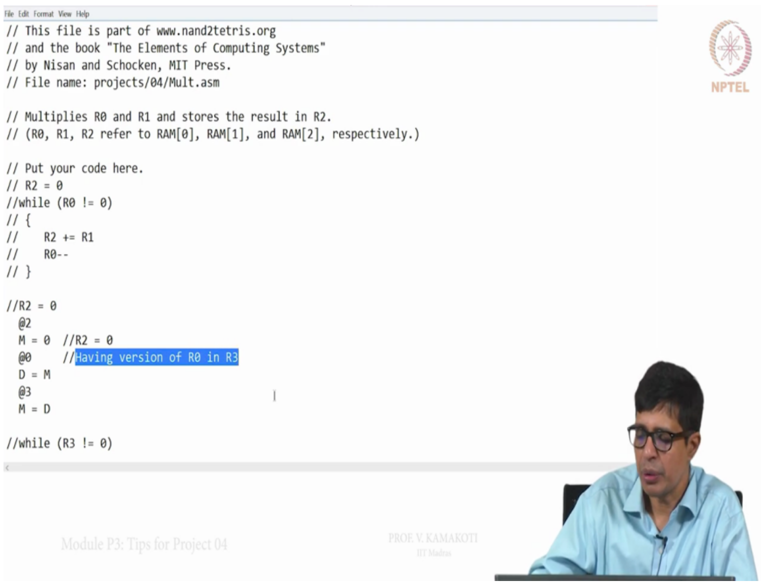
So how do we model multiplication using addition? So, we, so if I want to do A into B, I add A, B times. Add A, B times, this is essentially what we need to do, right. So that is precisely what we are going to do here, so let us look at the code very quickly. So, what I am going to do is, so this is the thing, I will copy A to R3 1st, because I do not want to destroy the value of A, I will create another temp location R3 wherein I will copy this value of A, right. And then what we do is, while R3 is not equal to 0, right, let R2 is equal to R2 + R1, R3 is equal to R3 -1.

Initially I will make let us say R2 is equal to 0, R2 is where I am going to write the sum, so initially I will make it 0 to start with and I will copy the value of A to R3, and while R3 is not equal to 0, I will keep adding R1. So, everytime I add R1 to R2, I will read, everytime I add R1, One-time R1, I will reduce R3. So the number of times R1 gets added to R2 is equivalent to A times. So I am adding A, B times, sorry, so I am adding A, B times, so this is how I can multiply, right.

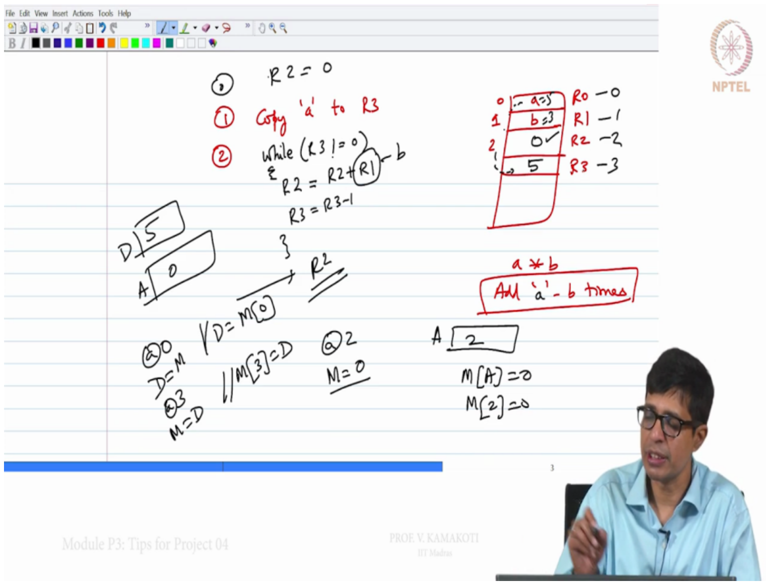
So 1st R2 is equal to 0, copy A to R3, while R3 is not equal to 0, keep adding R1 to this R2 again and again, how many times, R3 times, that is, that is stands for B, so keep adding B to

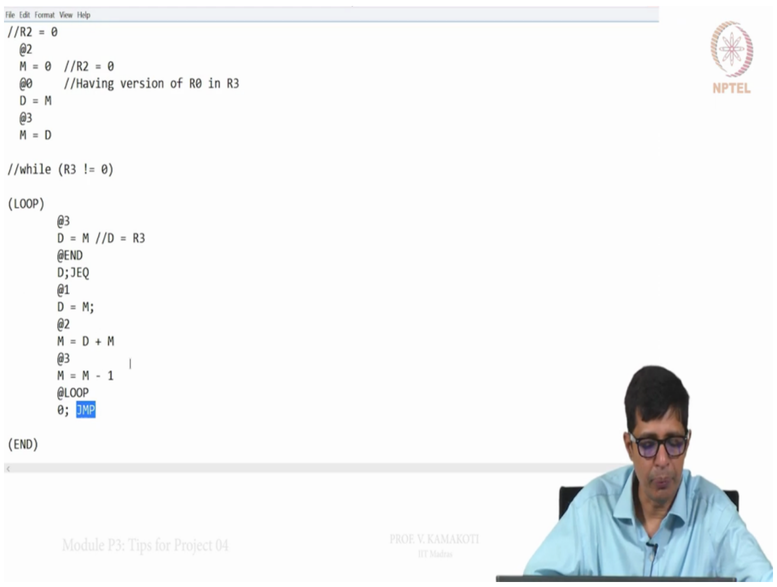
R2 A times. And this is the answer, so finally the answer will be in R2. Right, so this is the code and now with this code let us go and see how we are going to do this program.

(Refer Slide Time: 16:49)



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/04/Mult.asm

// Multiplies R0 and R1 and stores the result in R2.
// (R0, R1, R2 refer to RAM[0], RAM[1], and RAM[2], respectively.)

// Put your code here.
// R2 = 0
//while (R0 != 0)
// {
//    R2 += R1
//    R0--
// }

//R2 = 0
 @2
 M = 0  //R2 = 0
 @0     //Having version of R0 in R3
 D = M
 @3
 M = D

//while (R3 != 0)
```





```
//R2 = 0
  @2
  M = 0  //R2 = 0
  @0     //Having version of R0 in R3
  D = M
  @3
  M = D

//while (R3 != 0)

(LOOP)
        @3
        D = M //D = R3
        @END
        D;JEQ
        @1
        D = M;
        @2
        M = D + M
        @3
        M = M - 1
        @LOOP
        0; JMP

(END)
```

So the 1st thing is, the 1st thing that we need to do is make R2 zero, let us start looking from here, R2 is 0. So R2 is at RAM location 2, so the moment we say R2, so your A register gets the value 2. Now I say M equal to 0, M of 2 actually become 0, that is a 2nd location becomes 0, which is equivalent to R2 becoming 0. Now add 0, what is stored in R 0, A is story R 0. D equal to M means D equal to M of 0, so D gets a value of A at 3, at 3 now, the A gets, the register A gets the value 3.

When I say M equal to D, what is the address now, 3 is the address, so M of 3 will get D, that means 3 now gets the value of A. So I am copying the version of A in R3, so this is not precisely we have done here. 1st we made R2 S0, the next thing is I copied A from our 0 to R3. So how did I do R2 equal to 0, 1st I say add to 2, add to 2 means you are A register will get the valuable to. Now I say M equal to 0, M equal to 0 essentially means M of A is equal to 0 or M of 2 is equal to 0, so the 2nd location here, this is zeroeth location, 1st location, 2nd location.

The next step, copy A to R3, I said add 0, so your A register now becomes 0, now I say D equal to M of 0, D equal to M, which essentially means, which essentially means D equal to M of 0, so whatever, let us say A equal to 5 and B equal to 3, so D gets the value 5. So, already has the value 5 now. Now I say add 3, and 3 is pointed to R3, now I say M equal to D, that is essentially means that M of 3 is equal to D, so this gets the value 5. So I have copied a value of R0 to R3, right.

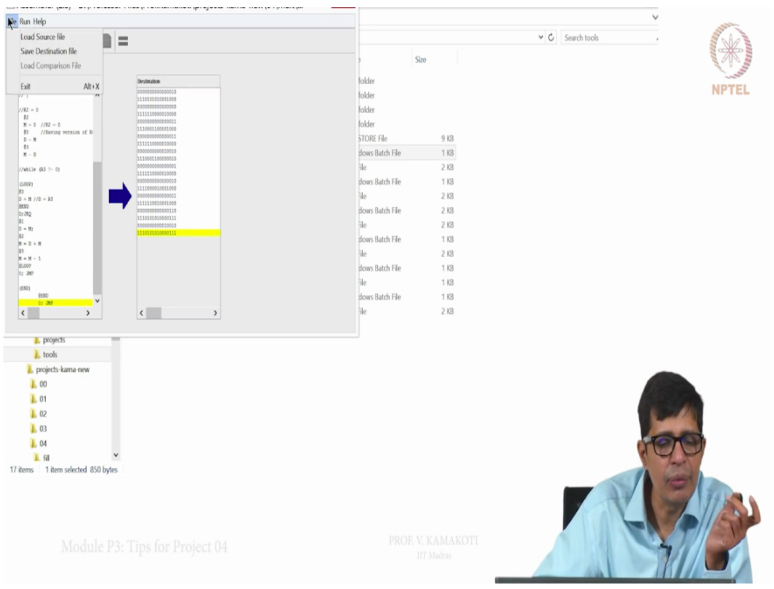
So the 1st 2 steps we have done now. The next step that we will do, we will just see here is as follows. So now we are going, so this is a loop. So what we did, while R3 is not equal to 0, what we need to do, while R3 is not equal to 0, R2 is equal to R2 + R1. So where is R2, while R3 is not equal to 0, 1st have to check that condition. 1st am checking the condition, 1st reading R3, where is R3 store, it is totally M3. So 1st time taking the address 3, I am copying the content of that address 3 into the D register, D equal to R3.

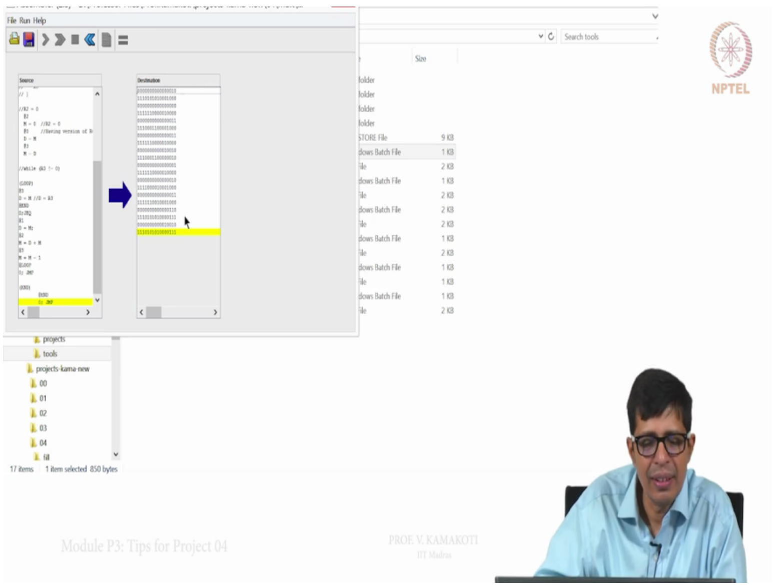
Now I am comparing D preservatives equal to 0, JEQ and when D is not, when D is equal to 0, that means I have to break this look, so I have to jump to end. So what I do at end, so the address register A gets the value of the symbol and I said D JEQ. D semicolon JEQ means what, if you are D register with a soaring R3 is equal to 0, the jump to the address in the A register which is D and D. So I just basically jumped to end if D equal to 0. So, while R3 is not equal to 0 is checked, this condition R3 not equal to 0 is checked here.

Then what I need to do? I need to do R2 in equal to R1 +, R2 + R1. So, 1st am accessing R1 at 1, so D equal to M, so D has the value of our work, then I say add 2, so now I say M equal to D + M. So I may not meet this semicolon, I may have the semicolon, does not matter. So M is equal to D + M. What is M equal to D + M, so what is D having, D is having the value of R1. Now I have put R2, so this M now is R2, this M is also R2. So R2 is equal to R1 + R2, right.

So I have now done that, then finally what I need to do, I have to decrement R3 at 3, M equal to M -1. After finishing it, have to start going back to this loop. So I put at loop here, so this is the start of this loop, and I do not check any condition, unconditional jump to loop, so I comeback here. So, by doing this we are basically done this part of what we are trying to do here. So while R3 is not equal to 0, R2 equal to R2 + R1, R3 equal to R3 -1, this we have coded.

(Refer Slide Time: 22:45)





So this is how your C program essentially gets translated to this. Now what we need to do is there are 2 interesting thing that are part of this whole stuff. There are 2 tools that are basically available, which you can use to verify whether your program is correct or not. So let us go into this. In your go to tools, now you can take what we call as the assembler. Click on the assembler and now we can open a file, the source file whatever you have taken, so this is there in your. Take this multi dot ASM file, so your file is here. Now we can basically see how you.

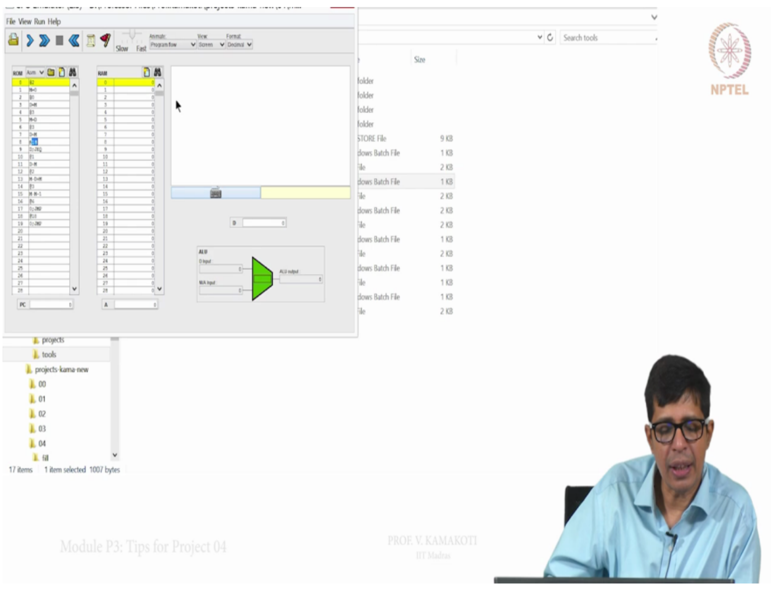
This is the assembler, so whatever you have written is in the assembly language or number next and that needs to be translated into binary. So this assembler basically takes your mnemonics and give the binary equivalent. See, how it is going to do. So the assembler has started working, so it, basically it is all comments it will ignore, as you see here, these are all comments. Okay, so this is the 1st statement. Add 2, add 2 is an A instruction, so this is a 16-bit size as you see here, the 1st bit is 0, and the remaining bit is 2, 0000010, so that you see here.

Next, next step, M equal to 0, M is a C instruction, so this is 111 and then the, this is A, A should be 0 and 101010 for C1, C2, C3, C4, C5, C6 essentially gives you 0 and then 001 is the destination, it is going to M, the destination bit and there is nothing, no jump here, so the last 3 bits are 0. So this is how you can interpret instruction by instruction. So we have seen in module 3.5 how A instruction and C structures, there in coded, etc. So those things you can basically analyse here and see how each instruction maps onto the corresponding bit patterns of those instructions.

So let us go ahead and do this entire thing. So if you click on this fast forward, you will see the whole program getting compiled. Done. So the last part we have just put at end 0colon jump, if you see here, right. So this is a infinite loop, this is an infinite loop, just for it to come and stop here. So this is how, so your program is now, the assembler basically converts your mnemonics to basic machine instructions. This is what, whatever you are seeing here this is what your system can understand.

Now this file, whatever, you can save this file, so this is, so you save this file, it is called, this file is mult dot ASM, that you have coded, this file, this is called mult dot hack which the assembler has generated. Now, the next thing is, once you finish with a similar, now you can go and do this emulator, CPU emulator.

(Refer Slide Time: 26:31)



This is quite interesting as you see here, the CPU emulator here, you can now load a program here, so we go back to this load a program, so I can load this hack program here, click on load. So your program has basically come here, okay. So for example, okay, and note that all your symbols are also basically resolved here. This is an ASM version, we can see the binary version also over here, this is the binary version of the program. For readability I can make it as an ASM version.

So all your symbols like loop, end, etc., for example end is 18, so at and 0 colon jump was that, so this end now has become 18. Now you look has become 6, because your loop started at this point and it is 6 again, right, your end is 18 as you see here, your end at this point you add end, at end, which was coming here, right, when you are while loop condition is satisfied. So this is not done, now what we are assuming is that 0 and, so let us say I will make it as 5, I can go and read this is the memory, so I will make this as 3, 5 and 3 are here, right.

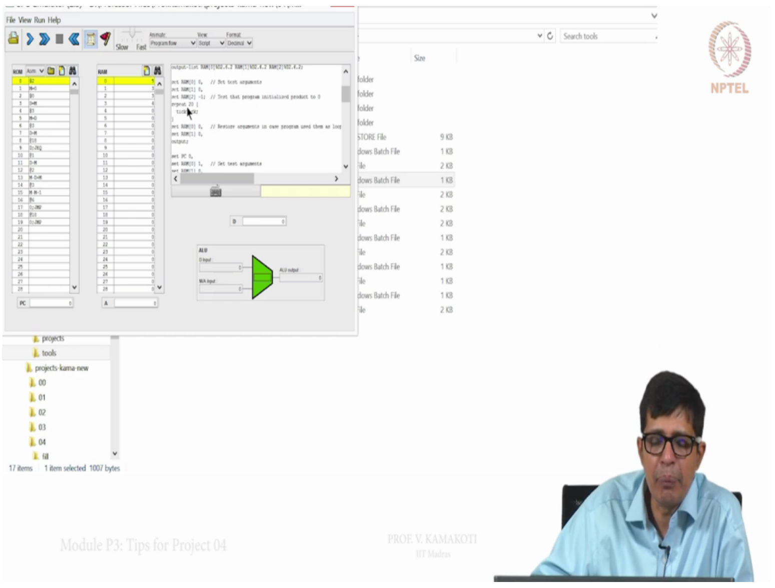
So now, now we will run this, we can actually make it a bit faster also. So this has come to this end loop and it is doing an infinite loop here, as you see here. But the answer that you got is 15, 5 into 3 is 15, okay. Now, so this is a verification that you program has actually done that correct. But now what we can also do is, so this is a slow-fast button which will see that your simulation can be, the feed of your stimulation can be actual animation can be faster or slower.

Now let us stop here, let us again go back here and we will start doing one by one. So, one of the interesting things that you should note here is this is the ALU, right. So 1st let us start the program again single step add 2, so we can say that, so this is over, so I can say stop. So, we will again loads of program, you, so now we will see what is happening add 2, so nothing happens at the ALU. Next is M equal to 0, so you M or A input is 2, and this becomes output, ALU output is 0, so now we are coming to add 0, okay.

D equal to, so next one D equal to M, okay. Just the instruction, the moment I do that instruction D equal to M, now you say that the D register actually becomes 5, right. So the ALU output became 5 and D register actually became 5. Now I am doing the next step up at 3. Now I say M equal to D, right, so the moment I say M equal to D, your M, the output is 5 and your D register is 5 and your location 3 also became 5. Now again I am accessing location 3, D equal to M and so your D register should become 5.

Now I am making at 18, D semicolon JEQ, no, then I am saying at 1 D equal to1, once I finish D equal to M, your D register should get 3 and so on. So if we can step through each one of these again and again and again, every instruction, what is happening to the memory, this is the memory part and what is happening at the ALU, you can basically see how the impact of each instruction on the memory and the ALU is and you can finish.

(Refer Slide Time: 32:36)



```
output-list RAM[0]%D2.6.2 RAM[1]%D2.6.2 RAM[2]%D2.6.2;

set RAM[0] 0,   // Set test arguments
set RAM[1] 0,
set RAM[2] -1;  // Test that program initialized product to 0
repeat 20 {
  ticktock;
}
set RAM[0] 0,   // Restore arguments in case program used them as loop
set RAM[1] 0,
output;

set PC 0,
set RAM[0] 1,   // Set test arguments
set RAM[1] 0,
```

Now, there is also another thing that we can load a script here, there is a script called mult dot test, we can load that script. Now, this is a script here, so when you look at this script, it says 1st load mult dot hack, compare, said RAM 0 is 0, RAM 1 is 0, make both as 0, RAM 2 is -1 and run the simulation. Similarly both are 0 and output, one of them has 1 and 0, one of them, 1 and 0, then 0 and 2, 0 into 2, then 3 into, then 2 into 4, so we can, and 6 into 7, 6 into 7.

So all different combinations this script will tell you and there are certain script arguments like set RAM 0 zero, set RAM 1 zero, RAM 2 , -1 and repeat 20 tick-tock means give a lot of clocks in between. So by the time this repeat completes, you are 0 into 0 would have been computed. Then it goes and restores the argument there and it checks the output. What is the output, output there is always a format, you output RAM 0, RAM 1 and RAM 2. RAM 0 into RAM 1 should be equal to RAM 2. So that is also given here.

So, so this script is quite easy to understand, right but now we can run the script. So if I say run the script. Now this has set RAM 0 and RAM 1 to 0 and now it is basically waiting for the stimulation to complete, right, repeat 20 times tick-tock. So, I can now make it a faster also. Now it has done 1 into 0 and it is just checking it 20 times. Then there is 0 into 2 and it is just checking it, 3 into 1 finished, 2 into 4 finished, 6 into 7, 6 into 7 again and it is done. So, basically I can run a script to verify this also. Right.

So with this we have given a demo of both the assembler and the CPU emulator. We will also do another example as module P4 that we will be presenting, where we will look at the interface to the I/O, basically the screen. It will do that example in the next module. Thank you.