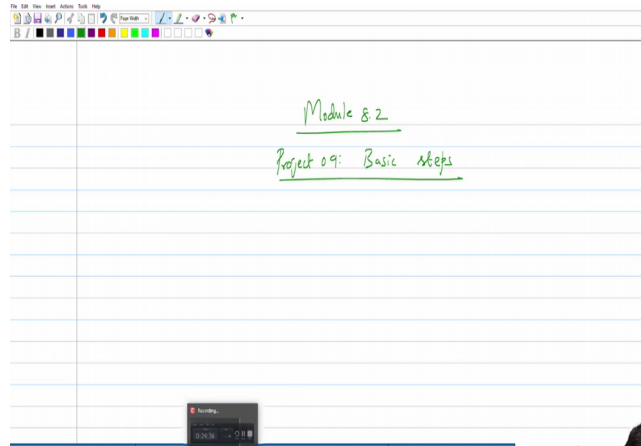



Foundations to computer systems design
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 8.2
Project 09: Basic Steps

(Refer Slide Time: 0:18)



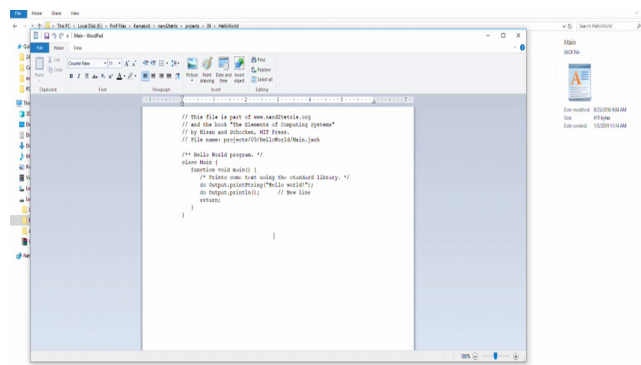

The image shows a digital whiteboard with a toolbar at the top. The text written on the board is:

Module 8.2
Project 09: Basic steps




Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras




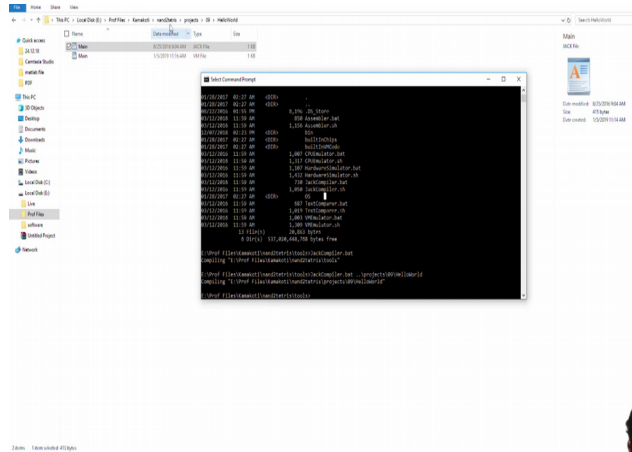
```
// This file is part of www.nptel.ac.in  
// and the book "The Elements of Computer Systems"  
// by Raman and Shobhan, MIT Press.  
// File name: project09/Module8/09a1-09a1.cpp  
  
/** Module 8.2 program - 1 */  
#include <iostream>  
using namespace std;  
  
int main() {  
    // Print out text using the standard library. ^V  
    cout << "Hello world!" << endl;  
    return 0;  
}
```



Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras





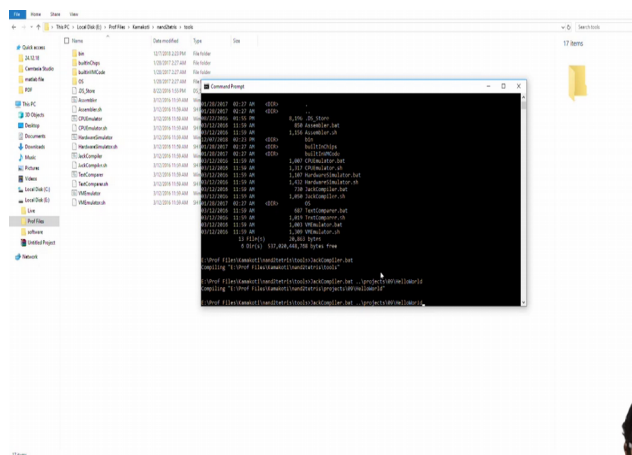
Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



So welcome to module 8.2, in this we will tell you about the project 09 basic steps, project 09 is one of the easiest of the projects, so what is there in project 09 that is go to the directories structure, so we go to nand to tetris, projects and we see 09, let us go and see the hello world, the hello world as a main dot jack, which is open with WordPad for just your understanding, so this basically has our hello world program as you see, now what we need to do is now we have to go back to our good world command prompt, if you are using Windows and then in the tools directory, nand to tetris tools directory.

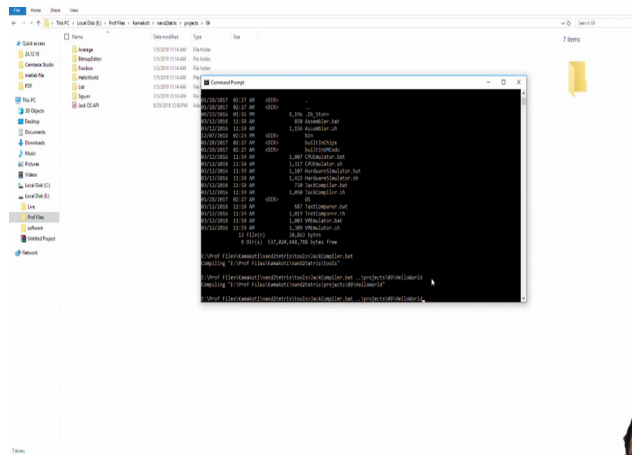
(Refer Slide Time: 1:22)



Module 8.2: Project 09: Basic Steps

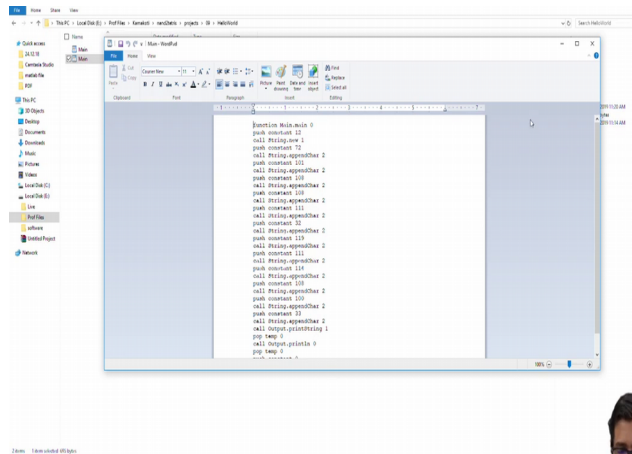
PROF. V. KAMAKOTI
IIT Madras





Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



So here when you go to nand to tetris tools and you actually have a Jack compiler, there is a Windows batch file and dot SH is the this thing, so we need to go to the command prompt and go to this same directory and their we can execute this statement jack compiler.bat.raj/project/09/hello world, so if you just see here the hello world is just a directory, so if you execute this statement as you see here .sh project/09/hello world all the jack files inside hello world will get compiled.

So I am just executing this command, so all the jack files inside this will be converted to M, so now after executing just see a main.jack, you just see a main.vm file, so there was a jack file now you will see vm file, now look at this vm file, so your entire jack programme has now got converted into vm right by the compiler, you have to write this compiler later but now we understand.

(Refer Slide Time: 2:51)

The screenshot shows a Virtual Machine Emulator window with the following components:

- Program List:** A list of assembly instructions including `mov rax, 0`, `mov rax, 1`, `mov rax, 2`, `mov rax, 3`, `mov rax, 4`, `mov rax, 5`, `mov rax, 6`, `mov rax, 7`, `mov rax, 8`, `mov rax, 9`, `mov rax, 10`, `mov rax, 11`, `mov rax, 12`, `mov rax, 13`, `mov rax, 14`, `mov rax, 15`, `mov rax, 16`, `mov rax, 17`, `mov rax, 18`, `mov rax, 19`, `mov rax, 20`, `mov rax, 21`, `mov rax, 22`, `mov rax, 23`, `mov rax, 24`, `mov rax, 25`, `mov rax, 26`, `mov rax, 27`, `mov rax, 28`, `mov rax, 29`, `mov rax, 30`, `mov rax, 31`.
- Registers:** A table showing register values for `rax`, `rbx`, `rcx`, `rdx`, `rdi`, `rsi`, `rbp`, `rsp`, `r15`, `r14`, `r13`, `r12`, `r11`, `r10`, `r9`, `r8`, `r7`, `r6`, `r5`, `r4`, `r3`, `r2`, `r1`, `r0`.
- Stack:** A table showing stack memory addresses and values.
- Call Stack:** A table showing the current call stack.

Below the screenshot, the text reads: "Module 8.2: Project 09: Basic Steps" and "PROF. V. KAMAKOTI IIT Madras".

The screenshot shows a Virtual Machine Emulator window with the following components:

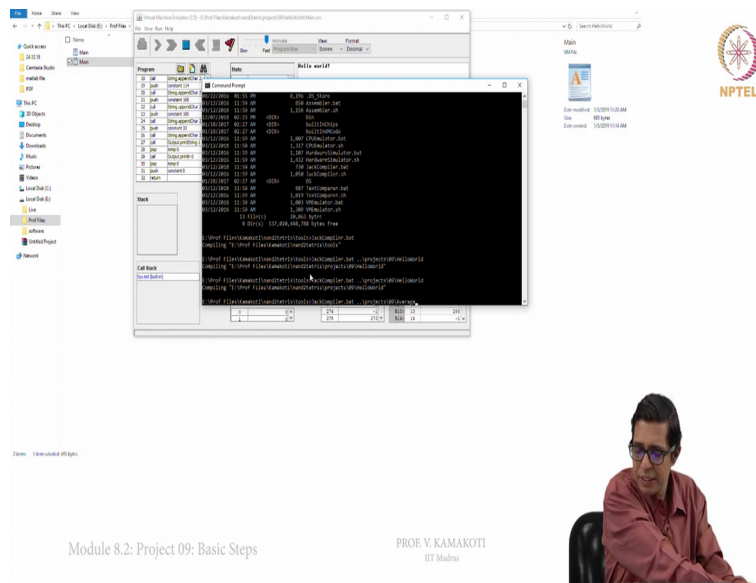
- Program List:** A list of assembly instructions including `mov rax, 0`, `mov rax, 1`, `mov rax, 2`, `mov rax, 3`, `mov rax, 4`, `mov rax, 5`, `mov rax, 6`, `mov rax, 7`, `mov rax, 8`, `mov rax, 9`, `mov rax, 10`, `mov rax, 11`, `mov rax, 12`, `mov rax, 13`, `mov rax, 14`, `mov rax, 15`, `mov rax, 16`, `mov rax, 17`, `mov rax, 18`, `mov rax, 19`, `mov rax, 20`, `mov rax, 21`, `mov rax, 22`, `mov rax, 23`, `mov rax, 24`, `mov rax, 25`, `mov rax, 26`, `mov rax, 27`, `mov rax, 28`, `mov rax, 29`, `mov rax, 30`, `mov rax, 31`.
- Registers:** A table showing register values for `rax`, `rbx`, `rcx`, `rdx`, `rdi`, `rsi`, `rbp`, `rsp`, `r15`, `r14`, `r13`, `r12`, `r11`, `r10`, `r9`, `r8`, `r7`, `r6`, `r5`, `r4`, `r3`, `r2`, `r1`, `r0`.
- Stack:** A table showing stack memory addresses and values.
- Call Stack:** A table showing the current call stack.

Below the screenshot, the text reads: "Module 8.2: Project 09: Basic Steps" and "PROF. V. KAMAKOTI IIT Madras".

Now we can execute this just for the it, now we can go here open up your vm emulator, it is just part of your thing, this is a vm emulator, vm machine immolate now what you see is you load program, there is hello world and you can load main.vm that you have generated here and it will say that we are using some of the inbuilt wise functions, like you are using `output.println` and `output.print string` etc.

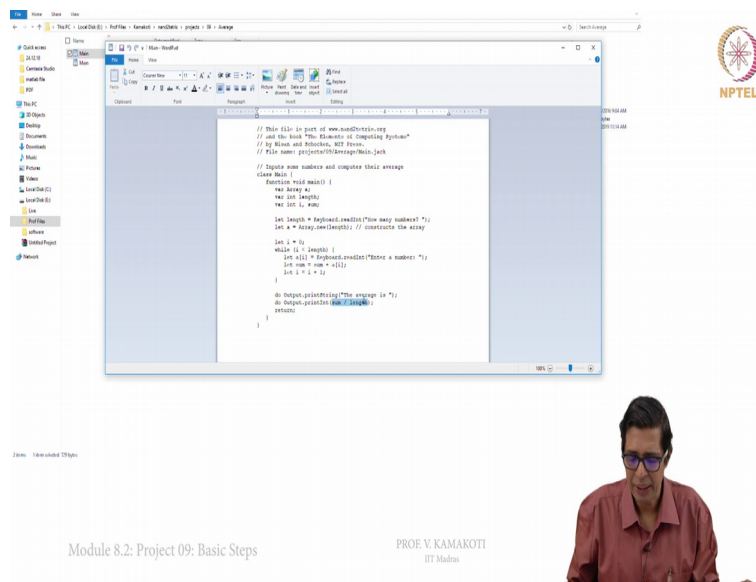
So you and you just run this program at full speed mode because it will be very slow, you just run this program, so this is the time taken for actually printing hello world on your screen, so what you see here is the screen I am just moving the mouse there, so you need to write a bit on that, my you see a hello world is coming up slowly okay this is back.

(Refer Slide Time: 4:11)



Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras

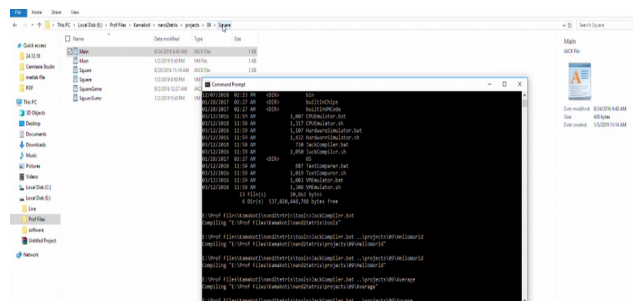
We will do one more here let us say jack compiler.bat projects 09 will do the average program, average this compile the entire average, let us go and see what is there inside average, inside the average folder, yes it is created the jack file basically except the number of numbers we want to add, we want to find the average and then it will take number by number and he was the average, the integer part of the average, sum by length, we have just seen this program.

Now let us see how we can execute this, let us go back to the, we have actually compiled it now we will just load that program in the jack emulator, let us go to 09, average yes, we load the main.vm here yes and that is it, so now we can see, run, this will take time and I am just waiting here, just to actually tell you that yes this this will take time and you need some amount of patients, at this point you just see that it is waiting, it is now come to the keyboard.readint, so this is asking you are many numbers, slowly yes.

So now we have to press the keyboard and say I want two numbers and then keep pressing the keyboard, note that I have press the keyboard symbol here and press enter their, now this is start working, so if you say I want four numbers than it will take 1 hour time, so just I have asked for two numbers, this is a emulation right, now it is asking enter the number yes, so it is now asking for enter number and its waiting now.

Now I say I am entering 2 and then I am entering number, now it last for the second number here yes, now let us say again I press the keyboard with my mouse and say seven and then I press the enter key also, now it should print my average as $7+2 \times 2$ should be 4.5, so 4 (()) (7:50) integer value, so it should print now the average is 4, so I suggest that you start executing this programs and understand how the vm later works, so this is done.

(Refer Slide Time: 8:11)



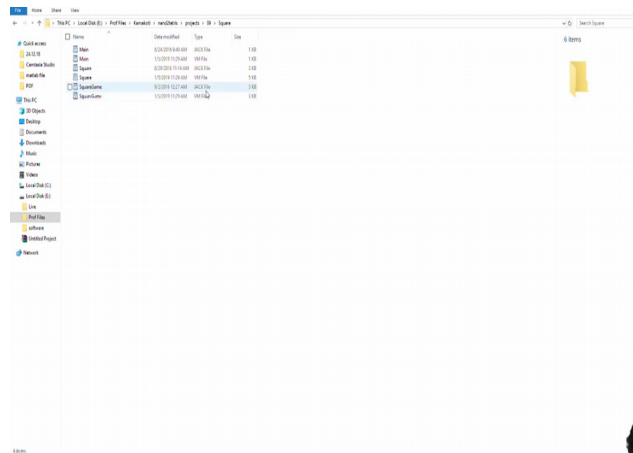
The screenshot shows a Java IDE window with a file explorer on the left and a terminal window in the center. The terminal displays the following code:

```
public class Main {  
    public static void main(String[] args) {  
        int sum = 0;  
        for (int i = 0; i < args.length; i++) {  
            sum += Integer.parseInt(args[i]);  
        }  
        System.out.println("Sum: " + sum);  
    }  
}
```



View: 1-embedded@80px





Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



Now let us see the next one right, will go back to the 09 project, now there is square function, now in the square function you will actually see, actually jack files, square games, square and main right, so this is a game where if you press the right arrow, the square will move right, top arrow it will move top, bottom arrow it will move down, left it will move left, so there is a square on the screen at the topmost, leftmost top corner and you can keep moving the square to your right, left, up, down using the arrow keys, so this is the basic thing.

Now there are more than 1 jack files here, 3 jack files so what will happen is initially, so the moment I go and execute square, /square I just use this name of this directory, so 09 has square, so I just go and execute /square, the moment I execute that all the jack files inside that gets compiled, so your main.jack became main.vm, the square.jack became square.vm, square game.jack became square game.vm.

(Refer Slide Time: 9:40)

The screenshot shows the Visual Studio Code interface with a C++ program open. A call stack window is visible, showing the current function and its callers. A confirmation dialog box is displayed, asking if the user wants to use functions not implemented in the RT SDK. The dialog box contains the following text: "No implementation was found for some functions which are listed in the RT SDK. The RT SDK does not provide built-in implementations for the C++ functions. If available, include the built-in implementations to use for functions which were not implemented in the RT SDK?"

Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras

The screenshot shows the Visual Studio Code interface with a C++ program open. A call stack window is visible, showing the current function and its callers. A confirmation dialog box is displayed, asking if the user wants to use functions not implemented in the RT SDK. The dialog box contains the following text: "No implementation was found for some functions which are listed in the RT SDK. The RT SDK does not provide built-in implementations for the C++ functions. If available, include the built-in implementations to use for functions which were not implemented in the RT SDK?"

Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras

The screenshot shows the Visual Studio Code interface with a C++ program open. A call stack window is visible, showing the current function and its callers. A confirmation dialog box is displayed, asking if the user wants to use functions not implemented in the RT SDK. The dialog box contains the following text: "No implementation was found for some functions which are listed in the RT SDK. The RT SDK does not provide built-in implementations for the C++ functions. If available, include the built-in implementations to use for functions which were not implemented in the RT SDK?"

Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras

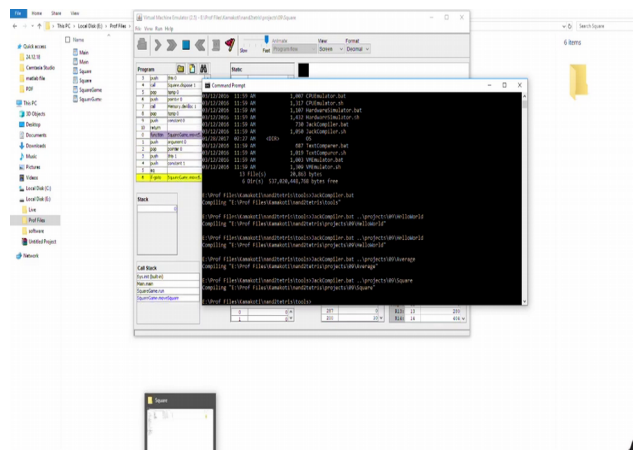
Now when we want to execute these now we go to this and in that point of time we just load program, in the previous case we are only 1 jack file 1 vm file in both the average and the hello world but when we go to the square, we have more than one, see we have more than one vm file, so what this software allows the vm emulator to load the entire directory, so when I say load the program the entire directory gets loaded, that means you are main gets loaded, main.main followed by square.new then square.raw all the functions get loaded here.

So the entire, so this is also what we call as linking at the virtual machine stage, so we had three programs, the main, the main was calling something call square, the square was calling something call square game or main was calling square game, square game was in terns by calling square, so be compile all the three and we load all the files in the directory, so how do we load again I repeat that you just click on the directory name, on the directory do not go inside the directory, we say load the directory itself, so again all the programs actually get loaded.

Now we can start executing this and slowly this will create a square yes, the square is basically printed, so I will go and say we want to move it right, so press the right arrow once, you see that it has moved a bit on the right, now we can still, so you have to click the mouse and keep, they take quite a bit of time to do this animation on a virtual machine emulator but this is how this whole thing keeps running, so this is a square game, so I allow you to play this their convenience but then you can keep moving this square up, down but it will be very slow than but you can keep doing it right.

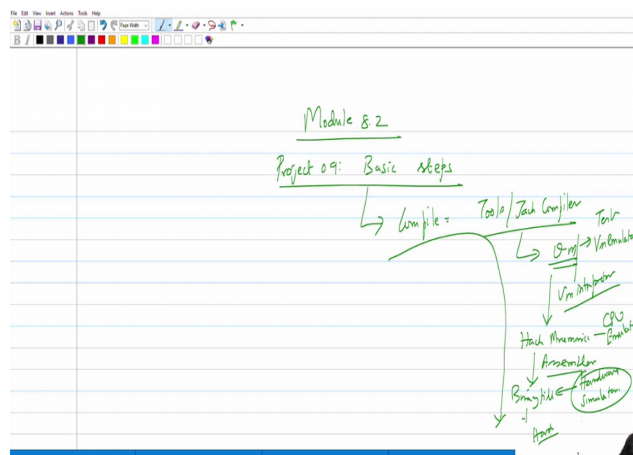


So what we have learned as a part of this is how to be compile the jack files using the jack compiler that is already provided to you right, so we will be writing our own jack compiler but now currently we are using a jack compiler, we have already had certain example programs, we have compiled them and to vm file and we are basically executing on the vm emulator.

(Refer Slide Time: 13:04)



Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



Module 8.2



Project 09: Basic steps

Tools/Jack Compiler → Object File → VM Interpreter → Hex Memory → CPU Emulator → Assembler → Binary File → App

Emulation
Simulation

Module 8.2: Project 09: Basic Steps

PROF. V. KAMAKOTI
IIT Madras



The other interesting exercise that you can also do is that now that you have created these vm files you can basically go use your own interpreters to compile take it to assembly file, you can use your same, you can use your assembler to make it into binary, so you can make, first you can make it into an assembly file and that assembly file you can put it on a CPU emulator and run and you can actually use your assembler to make it into your binary file and actually run on the hex emulator.

So all these things you can do, three levels of testing you can keep doing for as a part of this exercise okay, so this is what I basically wanted to our in module 2, I want all of you to basically execute this things, so compile using tools/jack compiler this will give you vm files, you can now test this vm files on the vm emulator, you can use your vm interpreter to get

your hack mnemonic files this you can run on your CPU emulator and see whether this is working, now you can use our own assembler that you have developed to get the binary file and this you can run on hardware simulator and see if it is working correctly.

So this entire set of things you can try and see if it is working and that will give you, already it will give you one thing, so once this entire stack your hardware is yours, your assembly is yours, assembler is yours, your vm interpreter is yours, only thing that is missing is the jack compiler and if you do that then you have understood how to build the entire system stack okay, so this is where, so kindly do this three programs and we will do many more as a part of project 09, couple of more programs as we proceed in this project. Thank you, so will see in the next module.