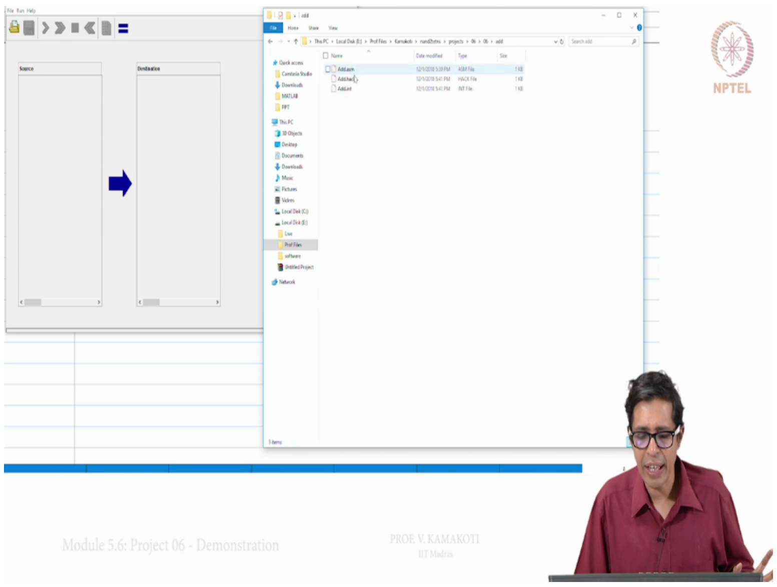
# Foundations to Computer Systems Design

**Professor V. Kamakoti**
**Department of Computer Science Engineering**
**Indian Institute of Technology Madras**

**Module 5.6**
**Project 06**
**Demonstration**

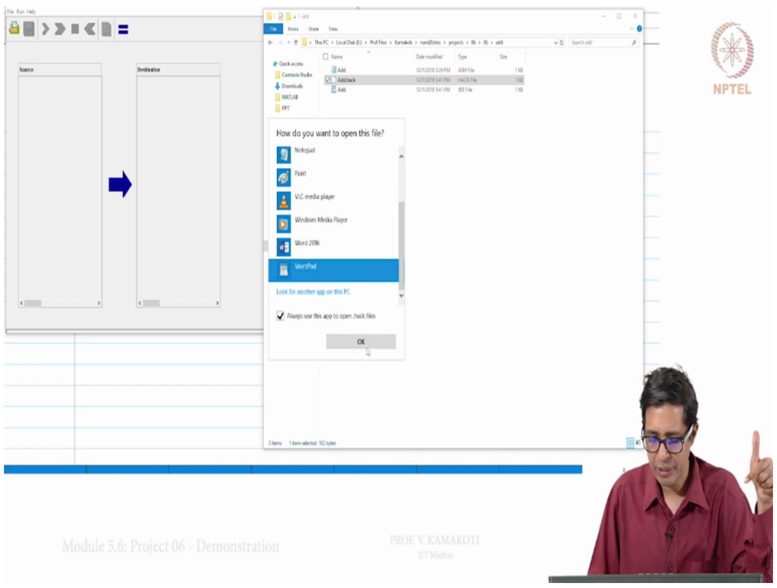
So welcome to module 5.6 and we will be doing a demo of you know project 06. We have written an assembler so let us just look into the directory of Project 06 so I am opening up this assembler.

(Refer Slide Time: 0:47)



So go to the tools and open your assembler here as I am showing here then then let us go to the projects so when you say this add so I have written the assembler, this add.asm will be already available to you as you see here, this is your add.asm sorry it should be in WordPad. This is your add add.asm, now we can see add.asm, I have generated this program that I have written the assembler has basically generated this add.int and there you will see you know, it is this notepad, we have to see the WordPad yeah right.

(Refer Slide Time: 1:51)


How do you want to open this file?

Notepad

Paint

VLC media player

Windows Media Player

Word 2016

WordPad

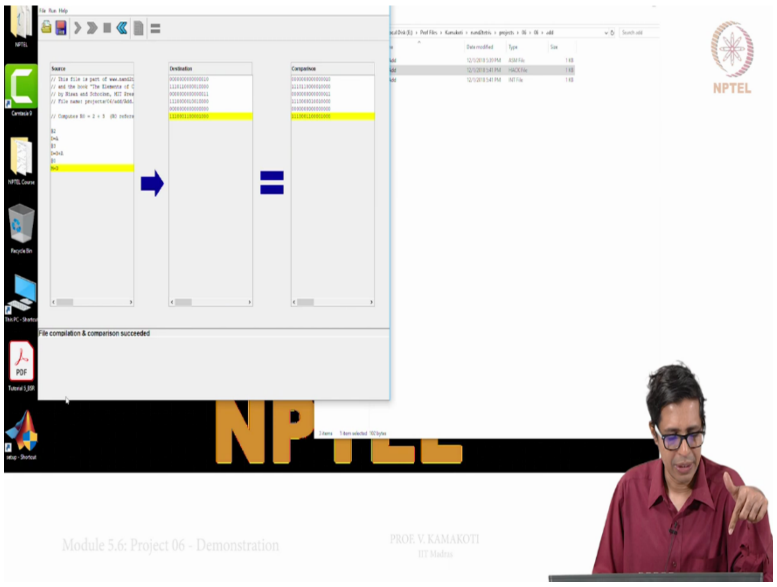
Look for another app on this PC

Always use this app to open .hack files

OK

Now all the comments are removed here so you can just see add, add was basically having the add had the comment and all those things are removed in this add.int so I have recommended add right, and then this as create as add.hac right so that again we will so that again these are 6 binary comments. Okay now how do we do this? So there is an automatic assembler that is provided as a part of your thing. Now in this automatic assembler and that is this will do the assembly for you what we do is first we load the file so I wrote add.asm, I also go and load the comparison file, this is add.hac that you have generated as a part of your family program.

(Refer Slide Time: 3:14)


Source

```
// This file is part of www.nand2t
// and the book "The Elements of C
// by Nisan and Schocken, MIT Pres
// File name: projects/06/add/Add.

// Computes R0 = 2 + 3  (R0 refere

@2
D=A
@3
D=D+A
@0
M=D
```

Destination

```
0000000000000010
1110110000010000
0000000000000011
1110000010010000
0000000000000000
1110001100001000
```

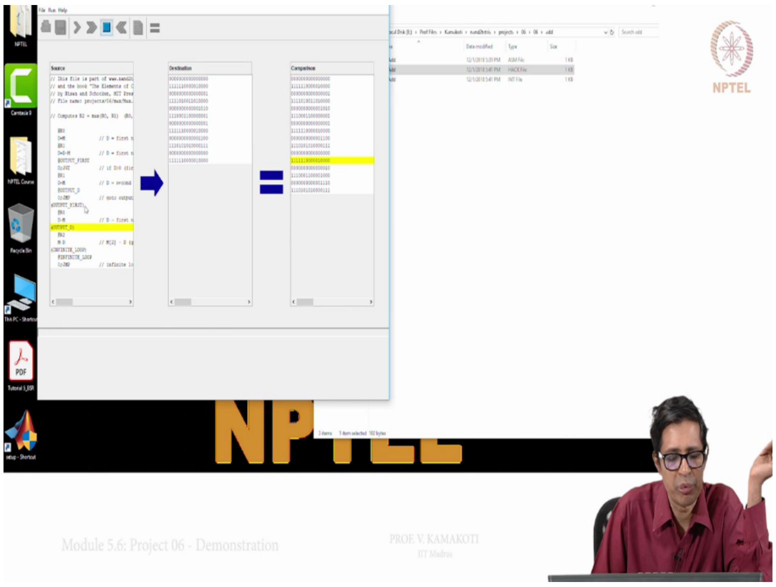
Comparison

```
0000000000000010
1110110000010000
0000000000000011
1110000010010000
0000000000000000
1110001100001000
```

File compilation & comparison succeeded

Now we will make this this assembler is already tested golden reference, now we will ask it to assemble so I type this so it is doing one by one and say yeah it is done, right. Now let us go and do something more, I can load another file now this add is, there are multiple directories in this 0 fix, we have added max, pong, etc, so let us take some max. Max has one with label and one without label, let us take this one with label, I have loaded the source file now I can also load the comparison file for max again root comparison file, now we do this, this double arrow will automatically do for you. Add.not is 0 so instruction by instruction this is assembling

(Refer Slide Time: 4:05)



Source

```
// This file is part of www.nand2t
// and the book "The Elements of C
// by Nisan and Schocken, MIT Pres
// File name: projects/06/max/Max.

// Computes R2 = max(R0, R1)  (R0,

   @R0
   D=M              // D = first n
   @R1
   D=D-M            // D = first n
   @OUTPUT_FIRST
   D;JGT            // if D>0 (fir
   @R1
   D=M              // D = second
   @OUTPUT_D
   0;JMP            // goto output
(OUTPUT_FIRST)
   @R0
   D=M              // D = first n
(OUTPUT_D)
   @R2
   M=D              // M[2] = D (g
(INFINITE_LOOP)
   @INFINITE_LOOP
   0;JMP            // infinite lo
```

Destination

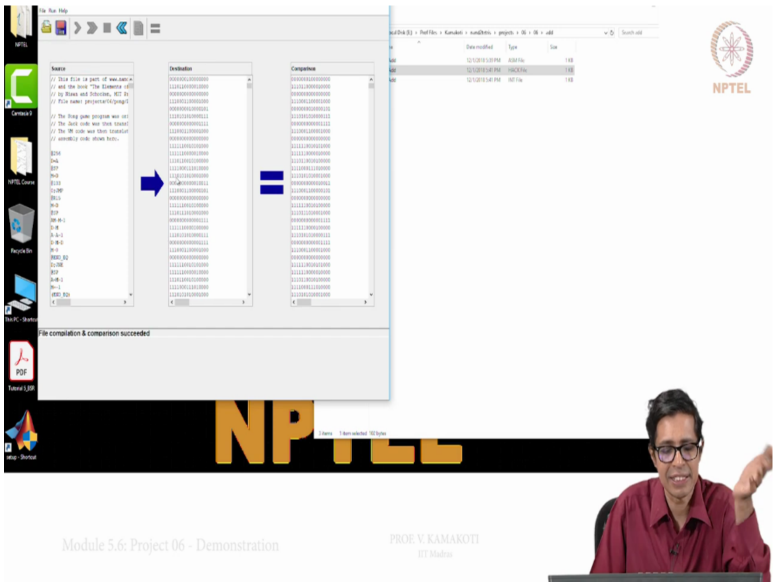
```
0000000000000000
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
```

Comparison

```
0000000000000000
1111110000010000
0000000000000001
1111010011010000
0000000000001010
1110001100000001
0000000000000001
1111110000010000
0000000000001100
1110101010000111
0000000000000000
1111110000010000
0000000000000010
1110001100001000
0000000000001110
1110101010000111
```

Note that output 1st was not… This is just a label and the comparison success so you can check this. So now next thing is let us go to some other file here like pong, pong is a very big program so let us take ponG with labels and let us take the comparison file again pong.hac I have created this, now let us do and this will take an enormous amount of time, there are 27,000 instruction so only tomorrow will finish. So now how we can do this? For running there is something called fast translation that is all it is done. so I will again do that. Note the source file, it is pong, load the comparison file which is again pong.

(Refer Slide Time: 5:13)


NPTEL
File compilation & comparison succeeded
Module 5.6: Project 06 - Demonstration
PROF. V. KAMAKOTI
IIT Madras

Now go to this png and say fast translation, there are 27,000 instructions and it worked, right. This is how you verify so you write the assembler take all the dotasm files in your in add, max, pong rect there are asm files, now you run your assembler on each of these files and that will give you this hac and intermediate and hac files similarly, for everything and then now you open this assembler which is part of your tools right. There is an assembler inbuilt assembler that is given to you, this is part of the tool and then go to each one of these files you load the asm files, you load as your source file and you also load a comparison file which is the file that you have created and then you go and run and ask it to you no fast translate or translate the base and see whether this matches, and this is the way you validate your assembler.

And if it works here, I think your assembler is perfect, so happy assembly and by the end of this I think you should spend 2-3 hours to get this assembler pucca, I can easily do that and that means you have written the 1st system program for your system stack, right. Now you have made a machine and you have made that machine more sensible by saying now at least I can give you a mnemonics environment. You can write assembly programs in pneumonic, this is more human readable and I have a software which will now convert it into binary and which can run on my machine. So this is the first step, I hope you are excited about this and now we will go into the next stage of understanding what the virtual machine is, right. Understanding a virtual machine is very interesting and I hope you will enjoy doing it.

So before we go to module 6, my request is please complete project 6; the assembler so that a complete understanding of the assembler is very important for you to appreciate what is your machine’s work, thank you.