Foundations to Computer Systems Design

Professor V. Kamakoti

Department of Computer Science and Engineering

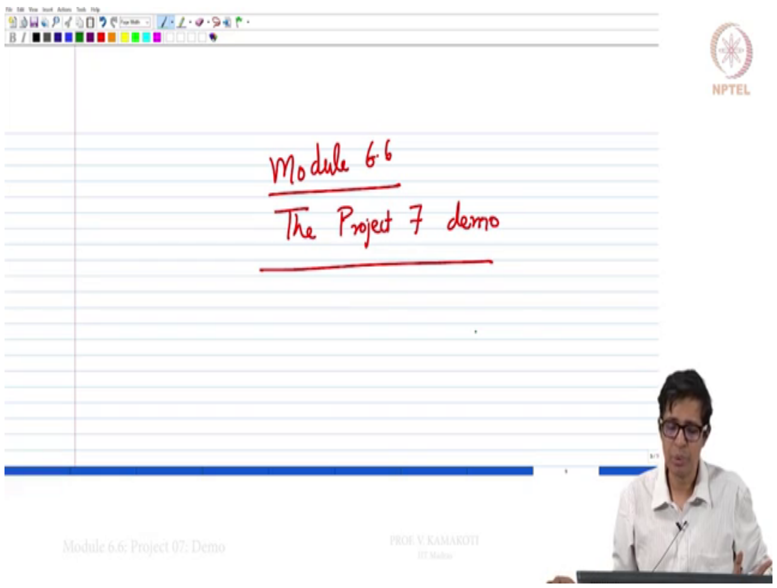
Indian Institute of Technology, Madras

Module 6.6

Project 7

Demo

(Refer Slide Time: 00:19)



Module 6.6

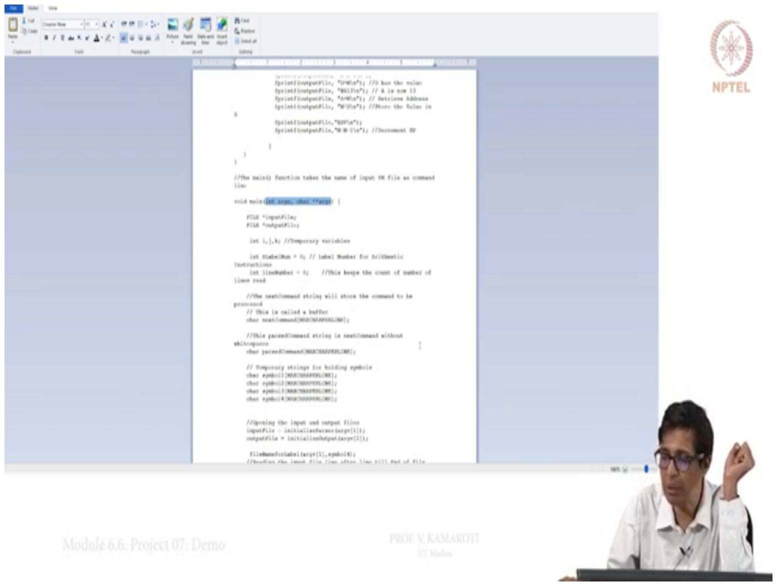
The Project 7 demo

Module 6.6: Project 07: Demo

PROF V KAMAKOTI

Welcome to module 6.6 and we do the project 7 demo before that I will show you the code for the VM virtual machine translator that we will see very quickly before we go on to the next stages.

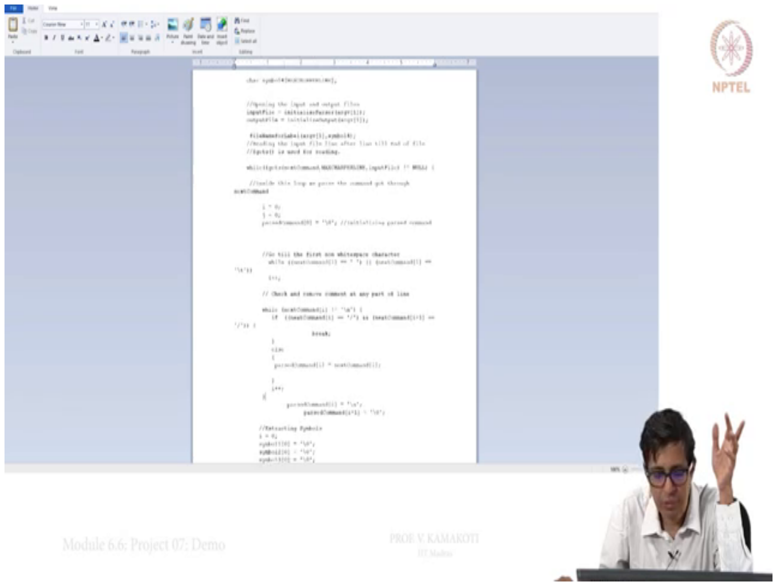
(Refer Slide Time: 00:32)



```
            fprintf(outputFile, "D=M\n"); //D has the value
            fprintf(outputFile, "@13\n"); // A is now 13
            fprintf(outputFile, "A=M\n"); // Retrieve Address
            fprintf(outputFile, "M=D\n"); //Store the Value in
D

            fprintf(outputFile,"@SP\n");
            fprintf(outputFile,"M=M-1\n"); //Decrement SP

        }
    }
}

//The main() function takes the name of input VM file as command
line

void main(int argc, char **argv) {

    FILE *inputFile;
    FILE *outputFile;

     int i,j,k; //Temporary variables

     int AlabelNum = 0; // Label Number for Arithmetic
instructions
     int lineNumber = 0;    //This keeps the count of number of
lines read

    //The nextCommand string will store the command to be
processed
    // This is called a buffer
    char nextCommand[MAXCHARPERLINE];

    //This parsedCommand string is nextCommand without
whitespaces
    char parsedCommand[MAXCHARPERLINE];

    // Temporary strings for holding symbols
    char symbol1[MAXCHARPERLINE];
    char symbol2[MAXCHARPERLINE];
    char symbol3[MAXCHARPERLINE];
    char symbol4[MAXCHARPERLINE];


    //Opening the input and output files
    inputFile = initializeParser(argv[1]);
    outputFile = initializeOutput(argv[1]);

     fileNameforLabel(argv[1],symbol4);
    //Reading the input file line after line till End of file
```

So the very simple code so I guess the taking a file and then through this so I will start with the main I will taken a file I have this is in (())(00:59) which we need, the input file is taken through argue and that input file whatever is start dot VM the output file will be start dot asm.

(Refer Slide Time: 01:21)



```
    char symbol4[MAXCHARPERLINE];


    //Opening the input and output files
    inputFile = initializeParser(argv[1]);
    outputFile = initializeOutput(argv[1]);

     fileNameforLabel(argv[1],symbol4);
    //Reading the input file line after line till End of file
    //fgets() is used for reading.

    while(fgets(nextCommand,MAXCHARPERLINE,inputFile) != NULL) {

     //inside this loop we parse the command got through
nextCommand

        i = 0;
        j = 0;
        parsedCommand[0] = '\0'; //initializing parsed command



        //Go till the first non whitespace character
          while ((nextCommand[i] == ' ') || (nextCommand[i] ==
'\t'))
          i++;

        // Check and remove comment at any part of line

        while (nextCommand[i] != '\n') {
           if  ((nextCommand[i] == '/') && (nextCommand[i+1] ==
'/')) {
                      break;
           }
           else
           {
            parsedCommand[j] = nextCommand[i];

           }
           i++;
        }
              parsedCommand[j] = '\n';
                   parsedCommand[j+1] = '\0';

       //Extracting Symbols
       i = 0;
       symbol1[0] = '\0';
       symbol2[0] = '\0';
       symbol3[0] = '\0';
```

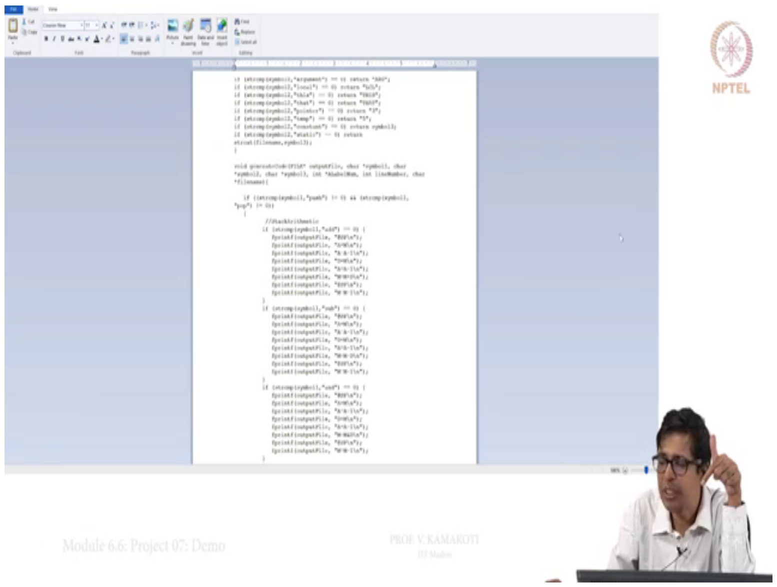
So I just do a single pass here so this is this main routine while F status I read line by line and I will initialize now variables first I do one scan till I reach my first character. So this particular thing is I remove all the white spaces that are before the first character.

So in the particular line which are read into the next command buffer right as in the case of your assembly I just go through till I get the first character and if the first character is not slash n it is not a new line I have not put or the first character is not a comment right then I keep taking this next command into a past command with till I reach the end of that line. So at the end of this particular while loop as you see here at the end of this first two (())(02:46) loop as you see here I have removed line that have comments if there are comments after the particular command if you have comments at that also I have removed and now I have the past command with the first character all these spaces before the first character are also removed in that line and any comments in the end of the line or at the start of line is (completely) removed.

Now I have the past command here now from this past command I extra three symbols namely symbol 1, symbol 2, symbol 3 so while past command is not equal to blank call not your missed command I just pass symbol 1 right, symbol 1 so symbol 1 there will be some white space then symbol 2 then white space then symbol 3 so this will extract the first symbol then from that point I will extract the second symbol and from that point I have also extract the third symbol. So at the end of this I have extracted three symbols from this and then know I if there is a symbol (()) (03:58) symbol 1 is zero or not then I generate this code I generate code for that particular one so I give the output file as an input this also this is a function called generate code to which I give an output file and I also give the three symbols, symbol 1, symbol 2, symbol 3 in each line and this is the A label and please note that I am passing it through reference.

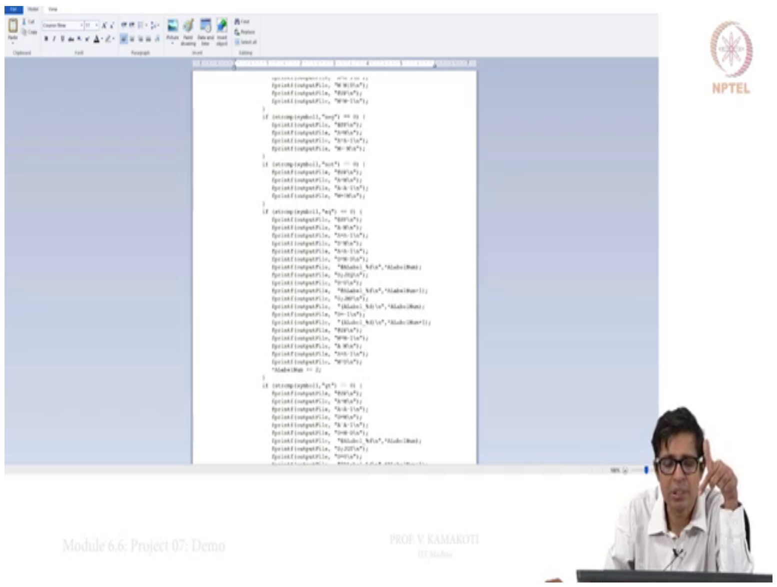
A label is the label that I want to generate unique labels that are necessary for this to generate in the case of the jump instructions then line number I use because don't forget that we need to have diagnosis always we need to have diagnosis so if there is something wrong we need to print the line number at that thing and I also have a symbol 4, the symbol 4 is nothing but you know for the static variable I need (())(4:59) for static variable I have to put as we said it is xxx dot index that xxx that first part of my file name suppose I am saying some add dot vm that add (a d d) should go in a symbol 4 this is for generating labels for the static variables as we have discussed.

(Refer Slide Time: 05:22)



```
if (strcmp(symbol2,"argument") == 0) return "ARG";
if (strcmp(symbol2,"local") == 0) return "LCL";
if (strcmp(symbol2,"this") == 0) return "THIS";
if (strcmp(symbol2,"that") == 0) return "THAT";
if (strcmp(symbol2,"pointer") == 0) return "3";
if (strcmp(symbol2,"temp") == 0) return "5";
if (strcmp(symbol2,"constant") == 0) return symbol3;
if (strcmp(symbol2,"static") == 0) return
strcat(filename,symbol3);
}

void generateCode(FILE* outputFile, char *symbol1, char
*symbol2, char *symbol3, int *labelNum, int lineNumber, char
*filename){

   if ((strcmp(symbol1,"push") != 0) && (strcmp(symbol1,
"pop") != 0))
   {
         //StackArithmetic
        if (strcmp(symbol1,"add") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "M=M+D\n");
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "M=M-1\n");
        }
        if (strcmp(symbol1,"sub") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "M=M-D\n");
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "M=M-1\n");
        }
        if (strcmp(symbol1,"and") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "M=M&D\n");
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "M=M-1\n");
        }
```

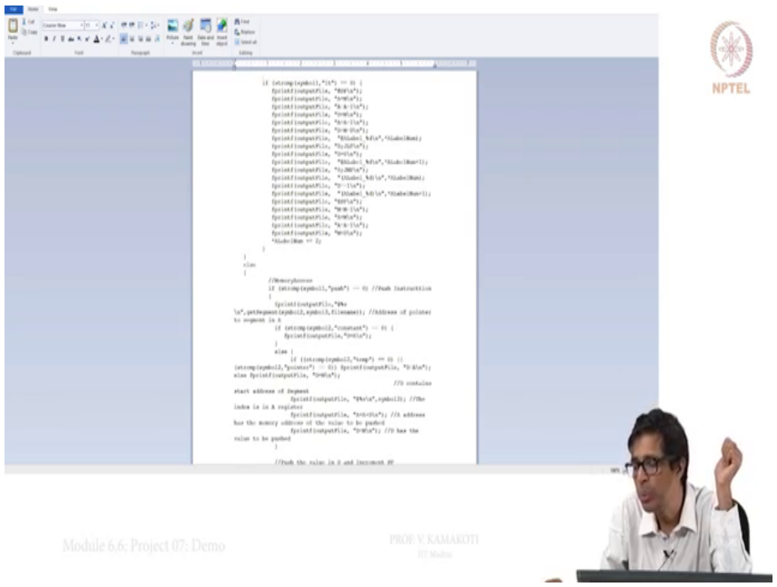
(Refer Slide Time: 05:48)



```
           fprintf(outputFile, "M=M|D\n");
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "M=M-1\n");
        }
        if (strcmp(symbol1,"neg") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "M=-M\n");
        }
        if (strcmp(symbol1,"not") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "M=!M\n");
        }
        if (strcmp(symbol1,"eq") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M-D\n");
           fprintf(outputFile,  "@Label_%d\n",*labelNum);
           fprintf(outputFile, "D;JEQ\n");
           fprintf(outputFile, "D=0\n");
           fprintf(outputFile,  "@Label_%d\n",*labelNum+1);
           fprintf(outputFile, "0;JMP\n");
           fprintf(outputFile,  "(Label_%d)\n",*labelNum);
           fprintf(outputFile, "D=-1\n");
           fprintf(outputFile,  "(Label_%d)\n",*labelNum+1);
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "M=M-1\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "M=D\n");
           *labelNum += 2;
        }
        if (strcmp(symbol1,"gt") == 0) {
           fprintf(outputFile, "@SP\n");
           fprintf(outputFile, "A=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M\n");
           fprintf(outputFile, "A=A-1\n");
           fprintf(outputFile, "D=M-D\n");
           fprintf(outputFile,  "@Label_%d\n",*labelNum);
           fprintf(outputFile, "D;JGT\n");
           fprintf(outputFile, "D=0\n");
```

Now the next thing is generate code, generate code is a simple routine here (())(5:29) so I take the first symbol I just compare it whether it is Push Pop, if it is not Push Pop then it is arithmetic then if it is add I put the code for add if it is subtract I put a code for subtract, AND, OR (()) (5:46)

(Refer Slide Time: 05:59)



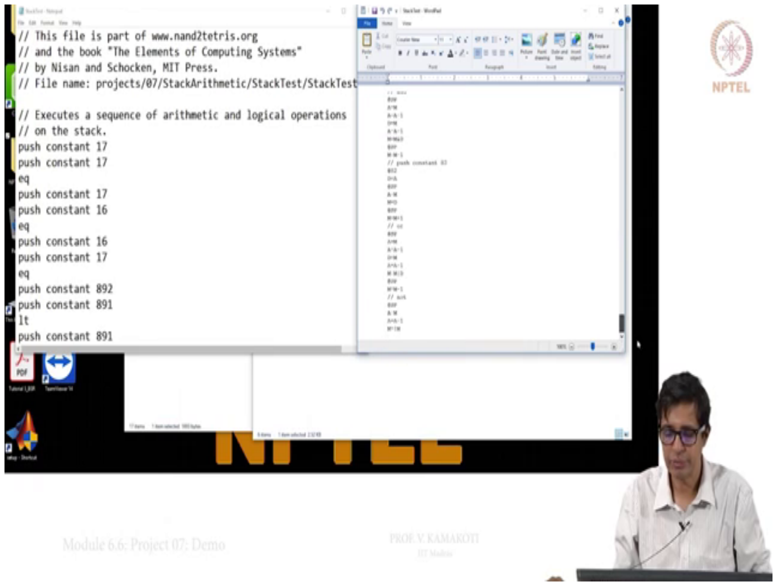
```
            if (strcmp(symbol1,"lt") == 0) {
               fprintf(outputFile, "@SP\n");
               fprintf(outputFile, "A=M\n");
               fprintf(outputFile, "A=A-1\n");
               fprintf(outputFile, "D=M\n");
               fprintf(outputFile, "A=A-1\n");
               fprintf(outputFile, "D=M-D\n");
               fprintf(outputFile,  "@ALabel_%d\n",*ALabelNum);
               fprintf(outputFile, "D;JLT\n");
               fprintf(outputFile, "D=0\n");
               fprintf(outputFile,  "@ALabel_%d\n",*ALabelNum+1);
               fprintf(outputFile, "0;JMP\n");
               fprintf(outputFile,  "(ALabel_%d)\n",*ALabelNum);
               fprintf(outputFile, "D=-1\n");
               fprintf(outputFile,  "(ALabel_%d)\n",*ALabelNum+1);
               fprintf(outputFile, "@SP\n");
               fprintf(outputFile, "M=M-1\n");
               fprintf(outputFile, "A=M\n");
               fprintf(outputFile, "A=A-1\n");
               fprintf(outputFile, "M=D\n");
               *ALabelNum += 2;
            }
      }
      else
      {
            //MemoryAccess
            if (strcmp(symbol1,"push") == 0) //Push Instruction
            {
              fprintf(outputFile,"@%s
\n",getSegment(symbol2,symbol3,filename)); //Address of pointer
to segment in A
              if (strcmp(symbol2,"constant") == 0) {
                 fprintf(outputFile,"D=A\n");
              }
              else {
                    if ((strcmp(symbol2,"temp") == 0) ||
(strcmp(symbol2,"pointer") == 0)) fprintf(outputFile, "D=A\n");
else fprintf(outputFile, "D=M\n");
                                                          //D contains
start address of Segment
                    fprintf(outputFile, "@%s\n",symbol3); //The
index is in A register
                    fprintf(outputFile, "A=A+D\n"); //A address
has the memory address of the value to be pushed
                    fprintf(outputFile, "D=M\n"); //D has the
value to be pushed
              }

              //Push the value in D and increment SP
```

Note I am using this A label and I incrementing it by 2, EQ and then greater than less than else this is going to be memory access so I put Push, if it is push then this part of the code will give me (the push) so one of the thing is when it is push I use this symbol 2 to basically get the I use the symbol 2 to get the get segment.

So if it is argument then I will return arg else here this that and if it is pointer I need to return three and I return five this is the segment code for (())(6:38) but if it is constant I have to return that index so I also take symbol 3 as an input to this and if it is static please note that if it is static I string at the file name is come here as symbol 4 along with your symbol 3 which is the index. So symbol 1 will be in the case of push command symbol 1 will be push, symbol 2 will be segments, symbol 3 will be index right the three symbols.

So it is file name dot that index will come, so this is what we do that in the case of static and then in the case of push we do this is the code for push, this is the code for pop and in the code for pop note that if your symbol 2 is a constant then it gives you an error stating the line number so this will help you to debug if you had a wrong vm code and so this basically is the code for pop and that is as per description as we had seen in the (past) ok.

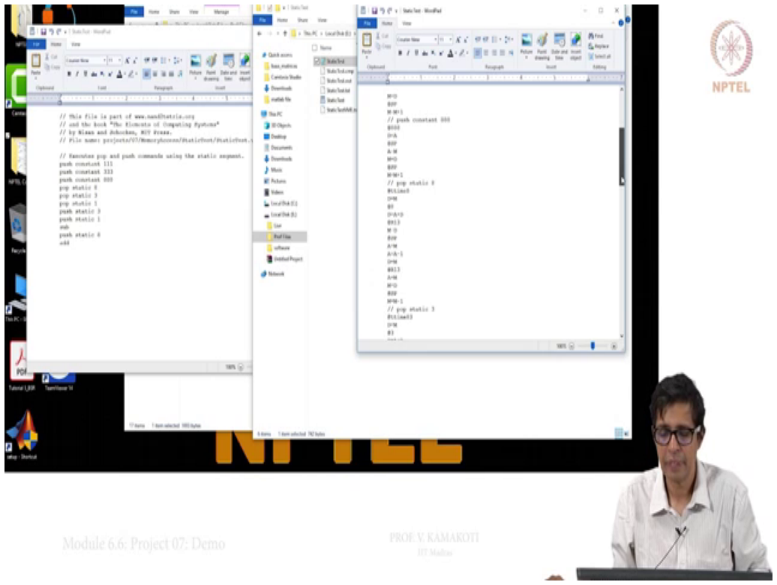
(Refer Slide Time: 08:21)



```
// 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/07/StackArithmetic/StackTest/StackTest

// Executes a sequence of arithmetic and logical operations
// on the stack.
push constant 17
push constant 17
eq
push constant 17
push constant 16
eq
push constant 16
push constant 17
eq
push constant 892
push constant 891
lt
push constant 891
```

So this is what we have done and let us now go and so to this code we actually give the input and will see the generated outputs, this is the input vm file and this is the vm file that has been (()) (9:29) file that has been created (sorry). So this is push constant 17 so as you see here push constant 17 this is the code for push constant 17, there is a code for EQ then this is a code for again push constant 17, push constant 16 and so on.

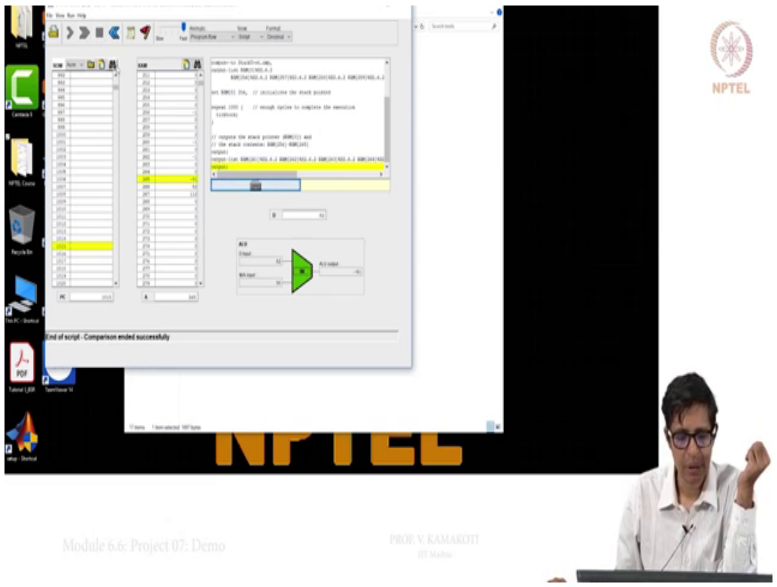
(Refer Slide Time: 10:19)



```
// 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/07/MemoryAccess/StaticTest/StaticTest.

// Executes pop and push commands using the static segment.
push constant 111
push constant 333
push constant 888
pop static 8
pop static 3
pop static 1
push static 3
push static 1
sub
push static 8
add
```

So will also see something on memory access, so this is the vm file and this is the asm file and you can see all this ok.

(Refer Slide Time: 11:20)



So now we will now go and show some execution here (sorry) so how do we do this execution? Now we go to this (())(11:38) we look at this seven we look at the tools we open his CPU emulator on the CPU emulator we can load this files I will just show you loading of one of this files so I have loaded this simple add now we can also load the script for that simple add (())(12:42) dot txt script we can load that and then we can run it.

Just run that so run it completely (make the stimulation faster also) (())(13:12) so you need to load the script for each asm file and then it will, so you can also do one more just to make your so we can write the script, load the script for we did for simple add now let us go for stack test (())(13:33) load the script here now run this right, ok. So compilation is successfully so this how we do this project and complete so this is the code.

So hope you will do project 7 very quickly as you see it is a very simple project so ant doubts please do put on the discussion forum and we can discuss it further. So now we move on to module 7 in the next session and that is going to be a very interesting module where we are now going to talk of the next two functions namely the program flow function and the program flow instructions and the function calling instructions and that will make our (())(14:51) complete, so before you go to there I want you to get a full understanding of this particular how this translation of memory and stack arithmaetic instructions work and that will make life much easier for you there right, so please spend that quality time and get this working, all the best, thank you.