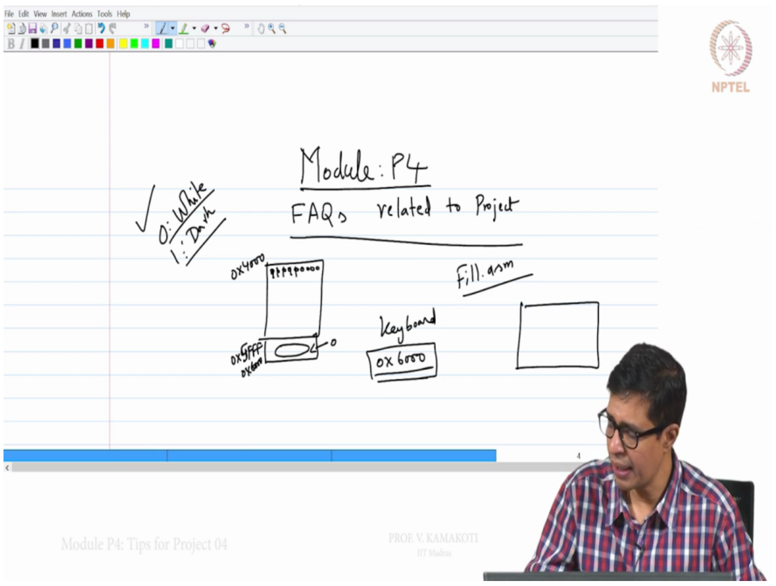
Foundations to Computer Systems Design.
Professor V. Kamakoti.
Department of Computer Science and Engineering.
Indian Institute of Technology, Madras.
Module P4.
Tips for Project P04.

(Refer Slide Time: 1:04)



So, welcome to module P4, this is again facts related to project. So, in the module P3 we actually saw how do we write assembly program to basically multiply 2 numbers, right. Now the the directory structure for the project P04 actually has another file that, another program that you need to write which is called fill.asm. So what is this fill program? In this, basically this program is to write, to make a write an assembly program in which you actually interface with your keyboard and screen, right.

So already we explained in the module 3 about memory mapped I/O, so if you want to communicate to the screen, you communicate it, the entire screen is mapped as 256 cross, 512 cross 256 matrix and basically so every, every pixel in that frame can be 0 or 1. 0 means it is white and 1 means it is dark. So the thing is that there is a memory, in this memory there is a space for every pixel, so this memory starts from 0 x 4000 and it goes to 0 x 5FFF, right.

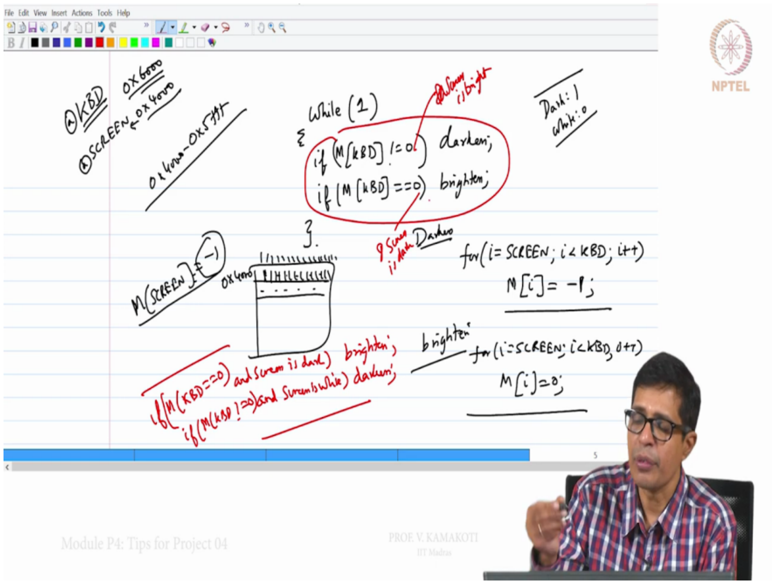
And for every pixel there is a bit out here. If that bit is zero, then the pixel is white, if the bit is 1, then the pixel is dark. Similarly for the keyboard, there is 0x6000, it is hexadecimal representation. There is one location, so whenever you press the keyboard, that value will be entered here, if no key is pressed, then the value is 0, right. So the fill program, what it does is it keeps on checking this 0x6000 location. Whenever it becomes nonzero, immediately it starts darkening the screen.

That is initially we assume that the screen is white, that is these values are all zeros, then the moment the program sees some value in 0x 6000, then it starts making it 1. That means your screen actually starts getting darkened. After it completes the darkening of the entire screen, again it will be, it will go and check the 0x 6000. If it is 0, if it is nonzero, the screen will remain dark, seat is completely dark now, then it will remain dark. If it sees a 0 here, that is no key is pressed, then it will start whitening the screen and so the screen will start becoming white.

So this is that fill dot asm. So, check the keyboard memory map, that is 0x 6000. If you actually find that it is 0, then the screen, you should start whitening the screen, if you find a nonzero, that is the keyboard is pressed, if the keyboard is not pressed you see a 0, if the keyboard is pressed, you start darkening the screen. And finish darkening, again check, then you again check, you find is nonzero, let it remain dark, when you see a 0, then you start whitening it. You whiten the screen, again go and check, if it is 0, then let it be white, if it is nonzero, then again start darkening it. So this is the program.

By writing this program, not only you will write assembly code properly, but also you will know how to interface with your keyboard and the screen, which is also important, these are the I/O devices. So the program, I am just going to give you the structure of this program. But then I will leave it for you to basically code it. We have actually, yesterday we took a C program in module P3 and we basically translated it to assembly. But now I will give you a sort of a C program that I expect all of you to code it in the assembly of hack, whatever assembly language we have seen so far.

(Refer Slide Time: 5:29)



```
While (1)
{
  if (M[KBD] != 0) darken;
  if (M [KBD] == 0) brighten;
}.

for(i= SCREEN; i< KBD; i++)
    M[i]= -1;

brighten
for(i=SCREEN; i<KBD, i++)
    M[i]=0;

if(M(KBD==0) and Screen is dark) brighten;
if(M(KBD !=0) and Screen is white) darken;
```

Okay, now what we need to do, so while, so there, let us, then the location called KBD, if you say at KBD, this KBD is equivalent to 0x 6000, this is where the keyboard input will be there. And then this at screen is 0x 4000, this is where the screen will start and from there it will go. So at KBD, while memory of KBD equal to 0, so we can write the code a little bit. While 1, that means this is an infinite loop, if memory of KBD equal to 0 , let us start with equal to 1 or actually not equal to 0, darken, okay.

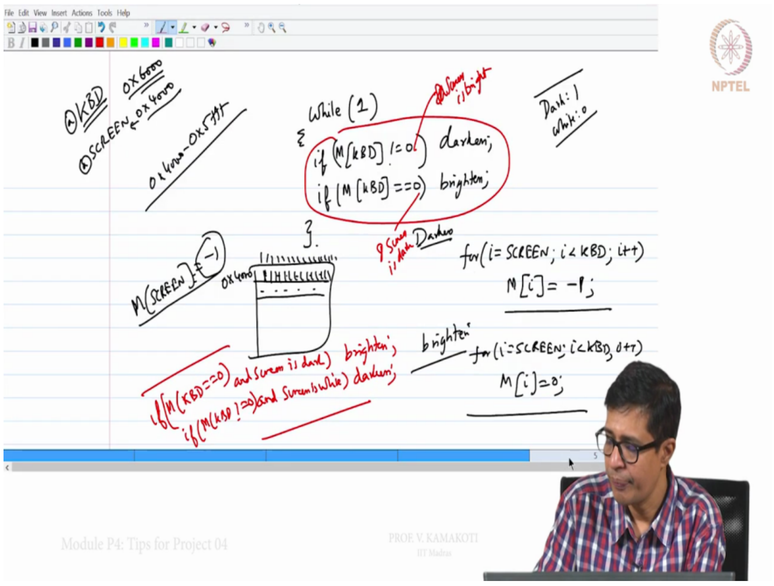
If memory of KBD equal to 0, brighten, this is essentially what we need to do, right. Now what remains by darkened? That means I have to write once, right because dark means 1, white means 0, so I have to write 1. So darken is for i is equal to screen, screen is 0x4000, i less than KBD because this goes from 0x 4000 to 0x5FFF, for i equal to screen, i less than KBD i + + M of i is equal to -1. What do you mean by M of i, this is basically darken, what is M, suppose I say M of screen is equal -1, that means the 16 bits, so every address has 16 bits.

So 0x 4000, this has 16 bits corresponding to 16 pixels on the screen and then next one 16 bits. When I say M of screen equal to -1, how is -1 represented in 2's complement, 111111111…, right, so it is represented as 1s. So, when I say M of screen is equal to -1, obviously this will all become 1 and it will start darkening. What is brightening? Brightening is for i is equal to screen, i Less than KBD i + + M of i is equal to 0, this is brighten.

So generally we need to do this part also, if, so what do you mean by, so if M of KBD equal to 0 and screen is dark, brighten it. If M of KBD not equal to 0 and the screen is white, darken it. Right, it is not that if M of KBD is not equal to 0, I were to say that screen is bright.

And when the screen is bright, then only I want to darkened. So I do not want to darken a darkened screen. And similarly when M of KBD equal to 0 and the screen is dark, then I want to brighten, okay.

(Refer Slide Time: 10:54)

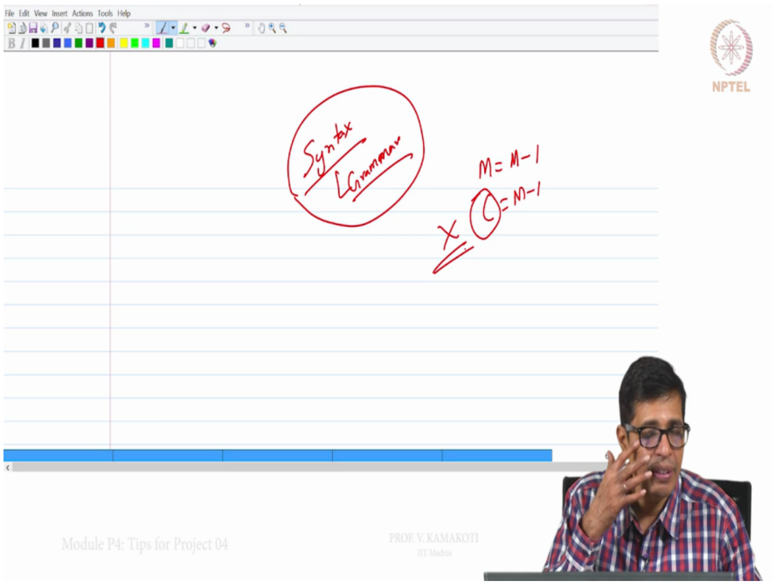


```
While (1)
{
if (M[KBD] != 0) darken;
if (M [KBD] == 0) brighten;
}.

for(i= SCREEN; i< KBD; i++)
    M[i]= -1;

for(i=SCREEN; i<KBD, i++)
    M[i]=0;

if(M(KBD==0) and Screen is dark) brighten;
if(M(KBD != 0) and Screen is white) darken;
```

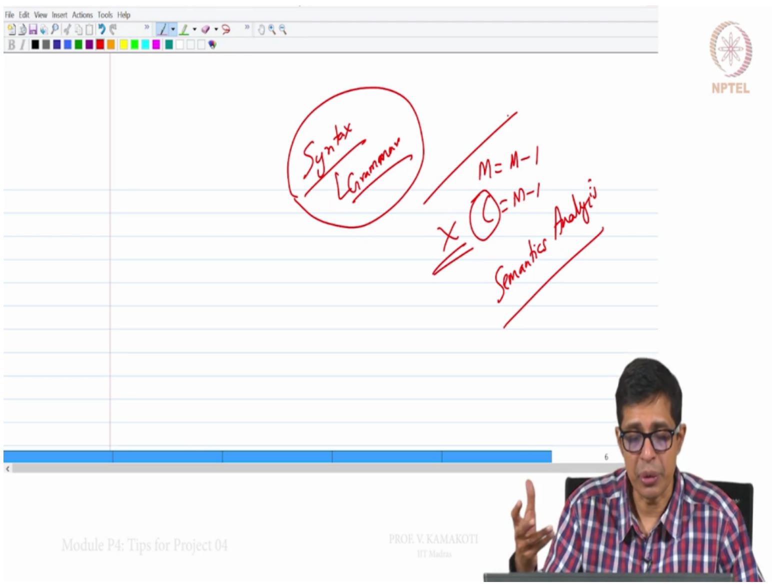


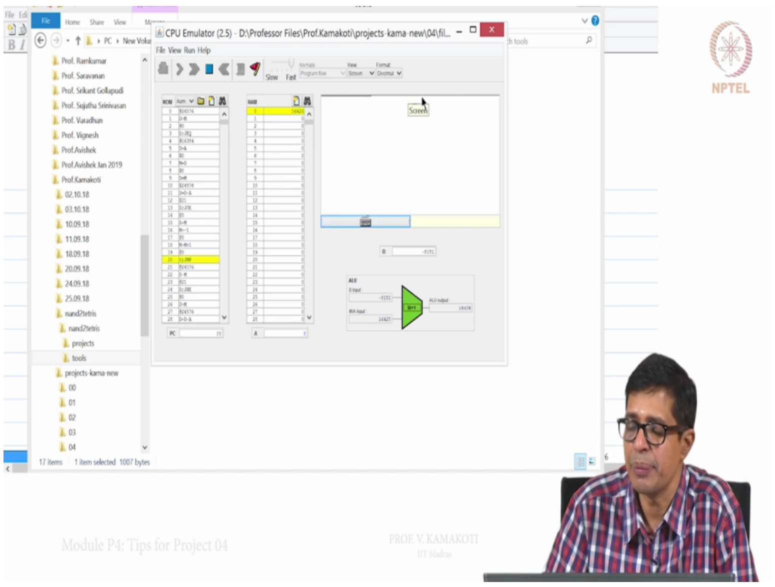
So this is a simple code, very very small code we can write and basically I will show you how it is going to work. So, we go to Project 04 fill and there just open up the tools, we will open up the assembler, open the load source files. So this is the source code, we will not discuss the source code, what we will just go and compile this. Now, this compilation process is nothing but the mnemonics get translated to the binary. So, this particular assembler converts the mnemonics into its corresponding binary, right.

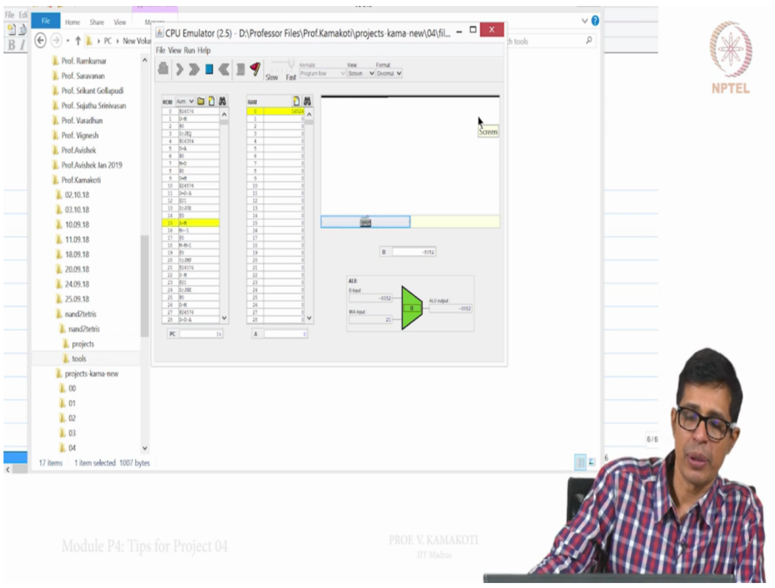
So this type of thing, what does this compiler, assembler do? It is a syntax analysis. The word that we use here is a syntax analysis. I am basically bothered, what do you mean by syntax, syntax is basically bothered about the grammar, whether it is followed currently or not. I am not bothered about the meaning as such here. So, have I coded, say for example M equal to M -1 is valid. Suppose I say C equal to M -1, this is not valid because we do not have anything called C in our machine language.

So there is a syntax error, syntax error in the sense there is an error in the grammar, I am trying to use alphabets which I should not use it. So there is nothing to do with the overall meaning of the program. What is the program trying to do, that we have not cared here. So, essentially when we do this compilation, this translation from the, from the pseudocode or mnemonic to this binary, we just do a syntax analysis. When we run this program, then that is where we are trying to understand the meaning of this program, that is called semantics analysis.

(Refer Slide Time: 14:11)




CPU Emulator (2.5) - D:\Professor Files\Prof.Kamakoti\projects-kama-new\04\fil...
File View Run Help
Screen
Module P4: Tips for Project 04
PROF. V. KAMAKOTI
NPTEL


CPU Emulator (2.5) - D:\Professor Files\Prof.Kamakoti\projects-kama-new\04\fil...
File View Run Help
Screen
Module P4: Tips for Project 04
PROF. V. KAMAKOTI
NPTEL

We will learn more about syntax and semantics when we go to the compiler but as of now just understand that we have, we indeed are doing a syntax analysis with the assembler while we are doing a semantic analysis when you take the CPU emulator. So let us take the CPU emulator here. So this is the, this is a screen and there is a keyboard also here. Now, what we are going to do now is will load our program. So we will load our fill program, fill dot hack, already there was a fill, this is loaded now.

So I could see, as I told you yesterday, I could also see the binary version or hex version or ASM version. Now this is how it works, now let us say that I am going to run this program and I can go and make it fast by doing this. Now the person is, that the program is actually waiting for an input, the screen is already white, this is the screen. Now how do I enter a keyboard input? With the mouse go and click this keyboard, click on this and put some say, some input, I am pressing Q, right, and, so Q has come.

So this particular program, (())(15:48), this is a buffer for the keyboard, it will shade Q, right. Now you see that it has started shading, you can see the screen becoming black. It will take quite some time for shading. Now you can see that the screen is actually becoming black. Right. And after it finishes, it again then comes and check the keyboard, at that point if you have not entered anything, immediately to start whitening the screen. So this is a simple program. I hope you can complete this program well. Right.

So with this you can safely finish off your project P4, so by this wave actually learned many things, we have learned how to do, right in assembly language, what is in a similar language, how to be right a code in a simple language, we have also seen the basic hardware that we have developed including the ALU. So this gives, this should give you an insight and working with the project also gives you that insight on the 1st few aspects of our system stack.

Namely the digital electronics, the microarchitecture and how this digital electronics and microarchitecture talk to the other layers, namely the operating system compilers and application programming. The insights that you will get from these 4 projects would be of immense benefit to you in your career. So kindly do all these 4 projects and if you have some queries, please do post in the discussion forum and I am sure one of us will help you complete these projects. Thank you.