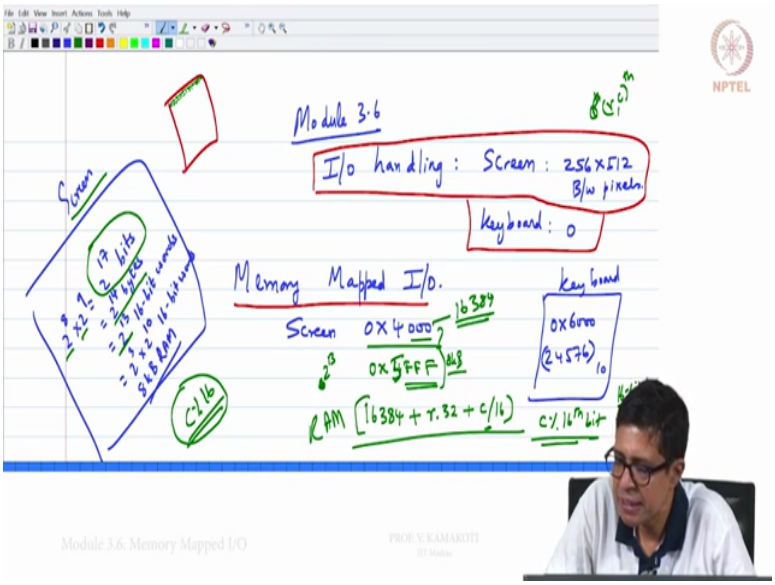
Foundations to Computer System Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 3.6
Memory Mapped I/O

(Refer Slide Time: 0:16)



So welcome to module 3.6, now we will be dealing (some) one more important aspect which is I/O handling, so when you design a computer namely this hack we need to have a method by which the user interacts with the system and that is through inputs and outputs.

So we left two simple input and output interface to hack one of the interface is screen to which will see the outputs and another interface is a keyboard through which will enter inputs to the system and we need to understand how these input and output are actually happening in real time system and for that they used a concept for foutycoles memory mapped I/O, that means whatever you want to output into the screen you write a tentacious specific memory and there will be graphics controller which will take from that memory and put it on to the screen.

So whatever you want to write on the screen you write it to some memory (and from memo) and from that memories some graphics some hardware will take it and show it on the screen, so as far as the cpu the hack that we are concerned we will just write in to memory and then we have some inbuilt screen controller which will go and show it on a screen, similarly even when you pressed a keyboard whatever the code that we pressed will come and reside in a

memory location and the hack can basically read from that memory location to find out what the user has actually entered, right.

So the way the hack talks to the (scr) communicates with the screen or to the keyboard is through memory locations so this is called memory mapped I/O. Now, for example now what we have is so the moment I have a I/O peripheral an input peripheral or an output peripheral immediately we assign certain memory address to that peripheral that you will see in many systems, right and where the we assign memory address essentially we go and say that when I want to read or write to that input or output device we essentially communicate through that memory.

So now let us take the screen, the screen that we have is 256 cross 512 black and white pixels means pixels is nothing but if you take a screen like this the screen can be broken up into small small small dots like that, (rows of dots) matrix of dots, so now we are looking at 256 cross 512 matrix the entire screen is divided into 256 rows and each row has 512 pixels (that is) and they are a black and white pixels, that means if it is zero that means that small part will be dark if it is one that small dot will be bright, so it is black or white, it will be dark or bright so you just see black and white pixels on your screen by lighting up certain pixels and darkening something you get some images and all those things, so the entire screen will be either that every dot can be made dark or white and you can get your desired photos or images or whatever you want on the screen.

Now, that means a screen is basically represented by a 256 cross 512 matrix of bits (so 256) so let us see some calculation here if I have a 256 2 power 8 true 512 is 2 power 9, that means entire screen can be represented as a 2 power 17 bits, that means 2 power 14 bytes, now in our memories that we have design in the sequential each memory has 2 bytes that is 16 bit words so we need 2 power 13 locations right, 2 power 14 bytes that means 2 power 13 locations because each location can store 2 bytes, so totally that means we need an 8 kb ram, so 8 kilo bit addressable each location word will store 16 bits.

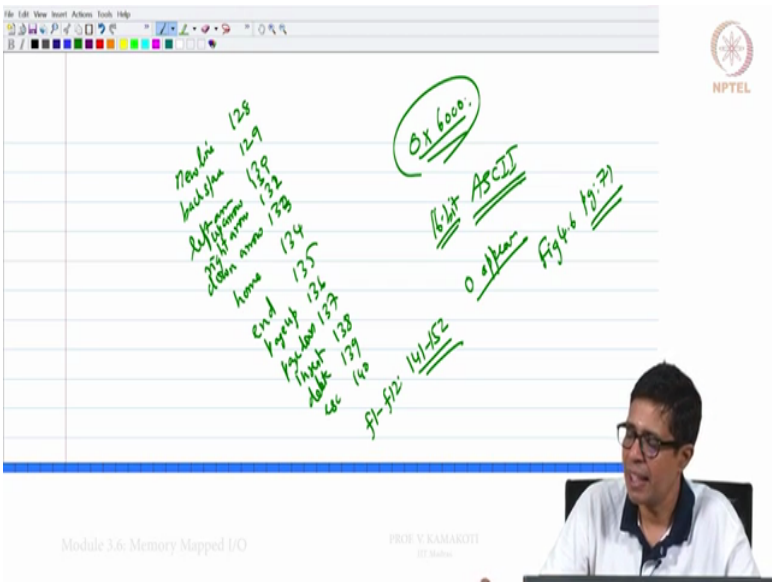
So if I have an 8 kb ram with each location storing 2 bytes each and 8 kb addressable then I basically can store this entire screen that, right. So (if I look at) your memory that is allocated for the screen starts at 0 X 4000 and from there 8 kb locations 8 kb is something like 2 power 13 right, so it goes to 2 power 13 locations from here, so 2 power 13 will be 0 X 4FFF right, (because to) so these is 4000 say 4FFF totally these is 8 kb no 5FFF right you can go up to 5FFF all the entire 8 kb can come here.

So 0 X 4000 is nothing but 16384, so if I want to find the in these matrix of 256 cross 512 bits if I want to find the C th if I say I want to find the r mass c th bit that r mass c th bit will be stored in the ram which is 16384 plus r row into 32 plus column divided by 16 these is the bit value to be stored, ok. So (you can in this) this is the address in this address you can take this whatever content is there so the c modulo 16 th bit in this address.

So this will give you in this address in this ram of this address you will get 16 bit number and you should take the (c mo) c remainder 16 that we should take. So this is basically so if you are given if you want r , c go to this address and in this address so you compute c remainder 16, this is c integer division 16 so go to this particular address get the 16 bits in those 16 bits (up to get) retrieve this c modulo 16 th bit and that will be this r, c th bit. So this is a very simple matrix calculation so we can do that.

So suppose I want to write into that bit go and set that bit that will become 1, if I want to brighten that bit make it 1, brighten the pixel make it 1 if you want to darken that pixel go and make it 0. So like that I can write every bit into this screen and the change whatever I want on the screen. So this is memory mapped I/O for the screen.

(Refer Slide Time: 8:38)



For the keyboard suppose the users presses something so (in the) there is a memory map 0 X 6000 allotted for the keyboard, so the moment user is presses a key then immediately what will happen is that the corresponding there will be corresponding 16 bit code the corresponding ASCII code, right. The corresponding ASCII code it is 16 bit ASCII code, ok right. The corresponding 16 bit ASCII code appears in the screen, right. When no key is

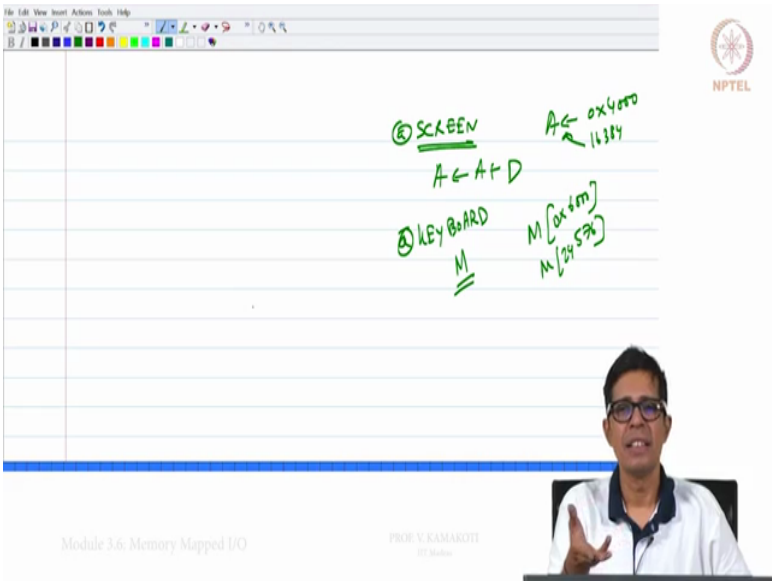
pressed 0 appears when no key is pressed is at 0 X 6000 when some key is pressed a 16 bit ASCII code essentially gets uploaded to the 6000.

In addition there are certain special keys which are in figure 4.6 of page 71 of the book very quickly we can see that there are you will have new line, back space, left arrow, right arrow, down arrow, home so this will corresponding code will be 128, 129, 130, 131, 132, there is an up arrow also which should be 131, 132, 133 and home will be 134 end page up, page down , insert, delete, escape so 135, 136, 137, 138, 139, and 140 and then f1 to f12 the function keys which will be 141 to 152.

So these special for the special keys that you see on the normal keyboard when you pressed that these are the values that will get entered on to 0 X 6000. So there is a keyboard controller which will basically understand which key of pressed and it will get that corresponding ASCII code or these special codes and these will basically go and load it into 0 X 6000. So at the processor hack when it once know what you are entered it will go to 0 X 6000 and it will find out what you have entered that and based on that it will start processing.

So this is basically how the basic keyboard that is the input handling happens, so I/O handling is also very important and so the way they have processor communicates with the device and the way the processor takes value from the keyboard we have just everything happens through memory and so this is called memory mapped I/O.

(Refer Slide Time: 12:12)



So one of the thing that (you) we need to see inside the program the moment I say ats screen then A register 0 X 4000 which is the value 16384 will be loaded into A register this is taken

care of. So then anything that you want to write into the screen we can do after this so if you just say M then M of A will be M of that screen or whatever of site so i can go and increment this A. I can make A equal to A plus 1 or whatever i can increment this and I can say A equal to A plus M A plus D, i can increment this and which I can access different aspects of this different pixel of the screen, right and then darken it or lighten it. So this is something possible.

Similarly i can say at keyboard and basically I can read the value. So the immediately I say M these is M of that 0 X 6000 which is M of 24576. So I can do both to basically read from the keyboard and write into the screen so this is support memory mapped I/O. So with this we come to and some understanding of how the basic instruction set architecture works and we looked at (three two instruction) two types of instruction namely A and C, A is very straightforward, C is essentially how we encode at C so that it maps on to our original value that we have designed, we saw a one to one correspondence between that and then now what we will do as a part of this project 4 that you need to be doing here we need to actually write some assembly code and see how it is (mapping on to the cpu) mapping on to the hack architecture.

So that project will be very interesting so very simple project we will be talking about that in the module P 3 that I will be following this, thank you.