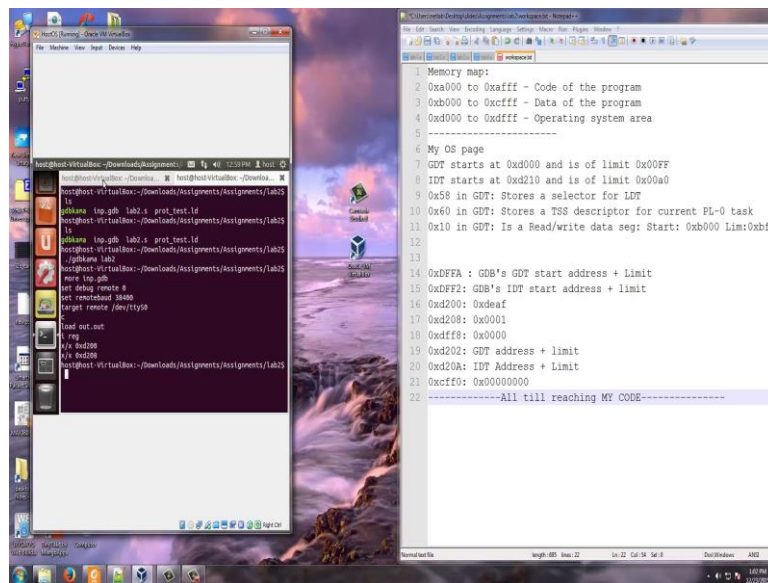


Information Security - II
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

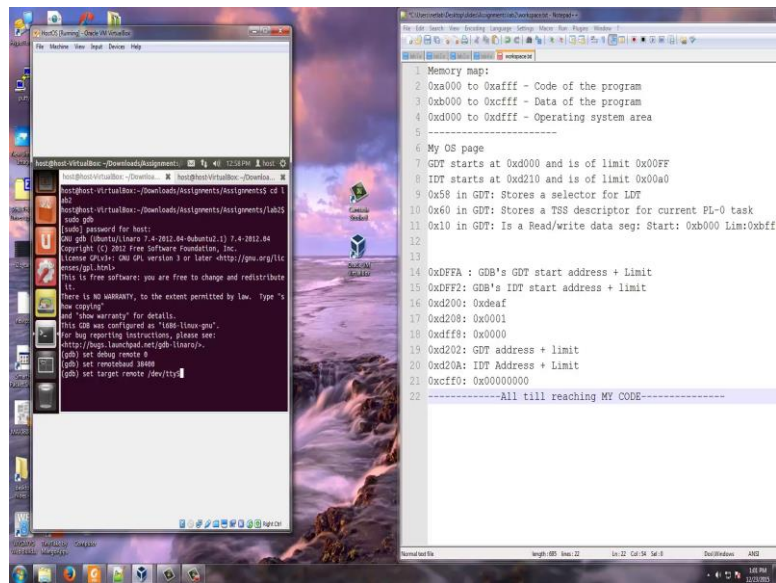
Lecture - 25
Lab2 Part 1 - Week 5

(Refer Slide Time: 00:09)



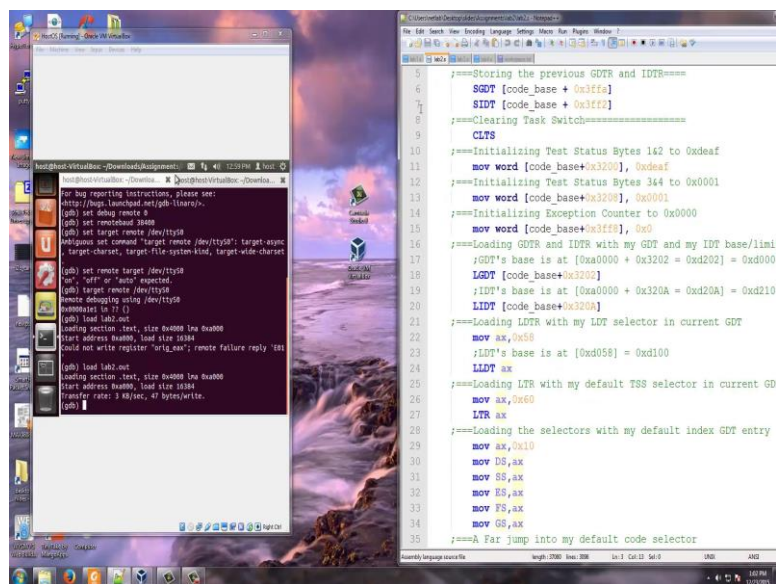
So, welcome to the second demonstration, where we are going to talk about the Interrupt Service Routine demo. Before we start on the demo let us do the basic setup. You have your virtual box opened; your virtual host is loaded. In the virtual host we have 2 tabs. Now, you have a terminal in which there are 2 tabs, one of the tabs go and compile, go to downloads assignments, assignments Lab2. In the previous demo, you are in Lab1 go to the Lab2 directory, now go and compile Lab2 slash slash GDB comma slash Lab2. Lab2 is compiled now.

(Refer Slide Time: 01:11)



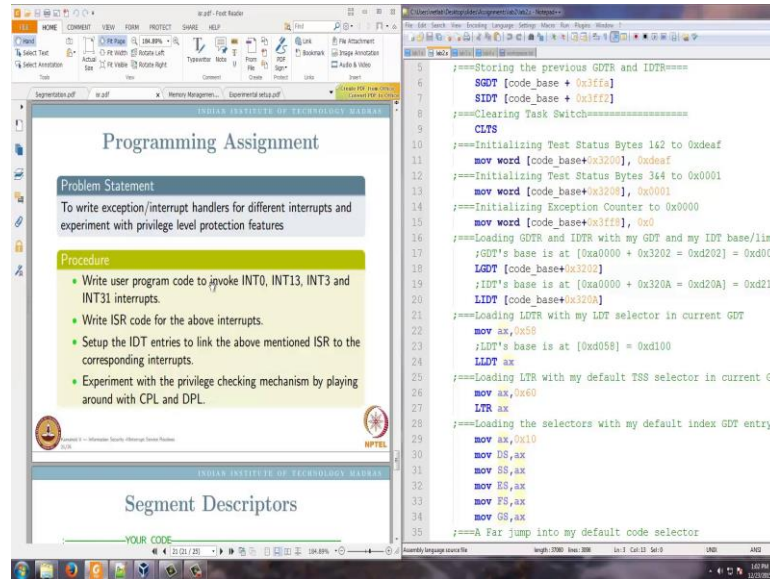
Go to the other thing; please note that this is also in Lab2 other tab. Now, we say sudo GDB give your password then again set debug remote 0, set remote board 38400, set target remote slash dev slash ttyS0 sorry set remote target. Let us see, what was that command, target remote?

(Refer Slide Time: 02:02)



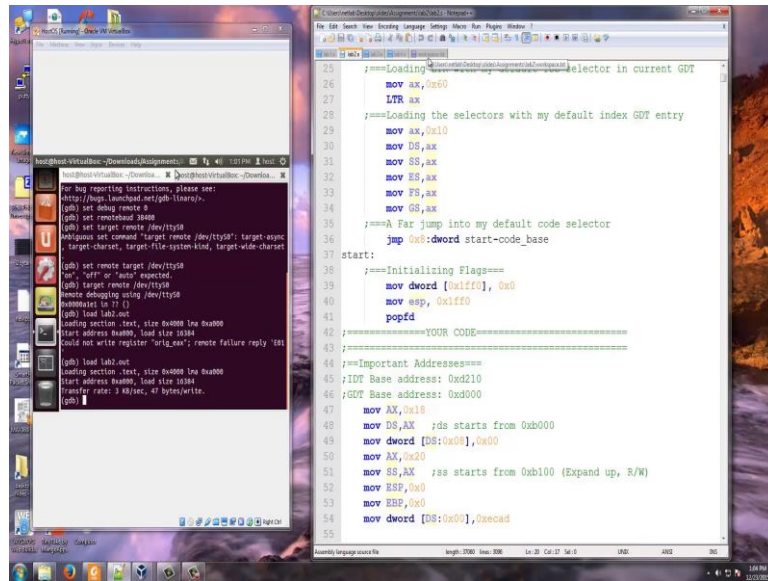
Target remote slash dev slash ttyS0. Now we will load the program Lab2 dot out. Again it is going to be loaded in a1000 as an error so again we load it, yes it is loaded. Now, let us go to Lab2 dot S here. So, before we proceed just let me give full demo of what we are trying to do.

(Refer Slide Time: 02:50)



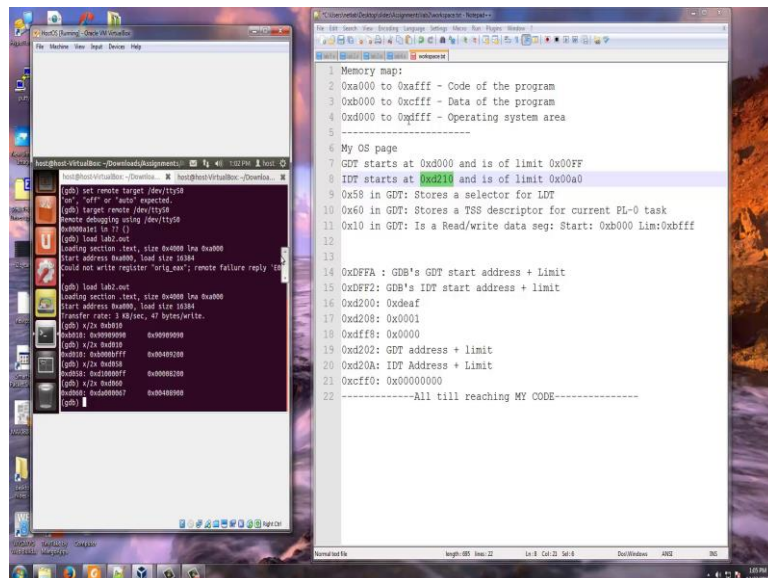
So, in this programming assignment, we want the user to write an assembly code to invoke divide by INT0 and INT4, which is not INT3, it is INT4, which is overflow exception and INT31, which is a software defined exception. INT13 is a general protection fault, so we leave it to you to get a general protection fault. However, in one of the later assignments we will also be creating a general protection fault but at this point we leave it as a simple exercise. We will generate scenarios where INT0, INT4 and INT31 occur and we will tell how the interrupt service routine basically works. And this is the entire thing that we are going to play on with this particular programming assignment. So we will go to the code directly.

(Refer Slide Time: 03:58)



Now, note that with respect to the programming assignment 1 and the assignment 2, please note that till 42, it was the same.

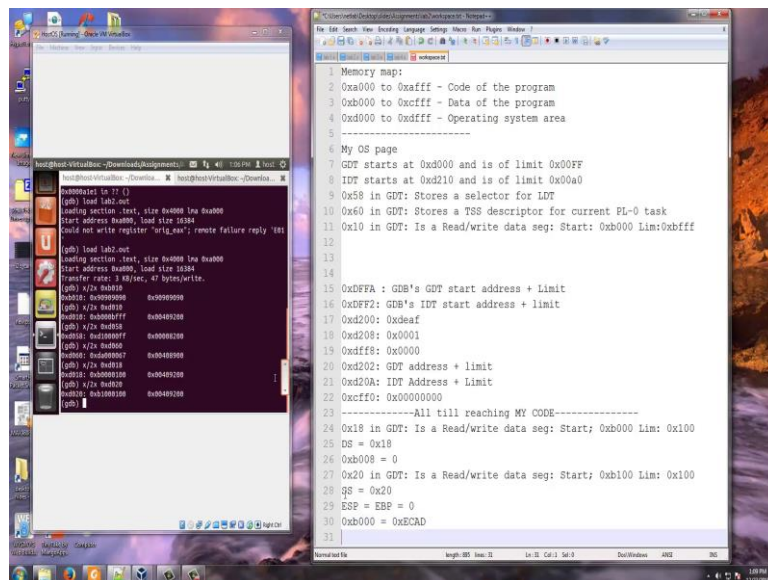
(Refer Slide Time: 04:11)



So what happened till 42, this is the memory map a000 to afff is where the Code of the program will stay, b000 to d000 the Data of the program, d000 to dfff is the Operating

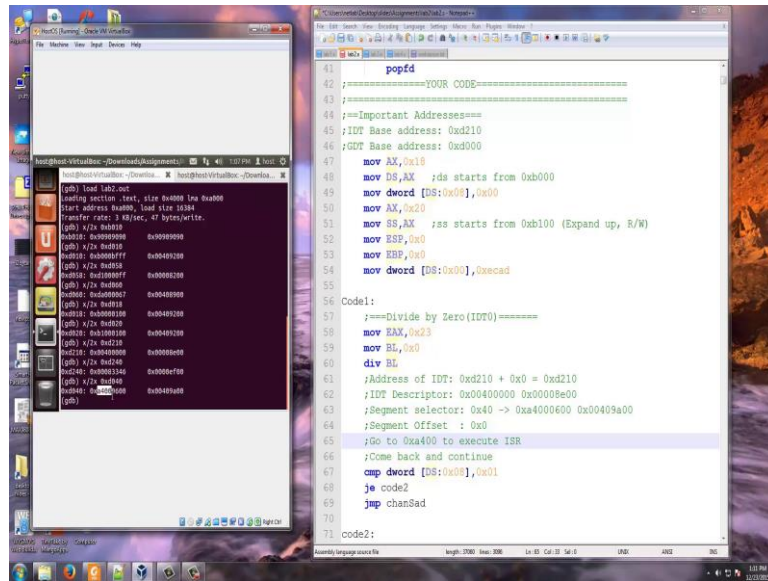
system area. Our GDT starts at d200, IDT at d210, 0x58 in the GDT actually stores a selector for LDT, 0x60 is stores the TSS descriptor, 0x10 in GDT is a read write data segment with 0xb000, 0xbfff. First just let us just check that x slash x 0 s slash 2x0xb 010, sorry b010, your b000 is rare and this is the segment b000. Similarly, x slash 0xd 058 selector for LDT and 0x60 is the TSS selector. Now, let us go and start working on Lab2 dot S. What we are doing now in this is the first thing is I am moving ax comma 0x18. So what is stored in x slash 2x0xb018? This b000 with 100 and type 2 segment is a read write segment.

(Refer Slide Time: 06:26)



So, your 0x18 in GDT is a read write data segment starting at 0xb000 with limit as 0x100 and that is load to DS. So DS is pointing to this. Now, let us go and see line number 49, so b008 should set to 0. So 0xb008 is set to 0 and mov ax comma 20, so what is x slash 2x0xd020 is b100. 20 starts with b100 of limit 100 and your SS now points to 20. Your ESP and EBP are 0, so ESP is made equal to EBP is equal to 0 and they have also made DS colon, so b000 is ECAD.

(Refer Slide Time: 09:30)

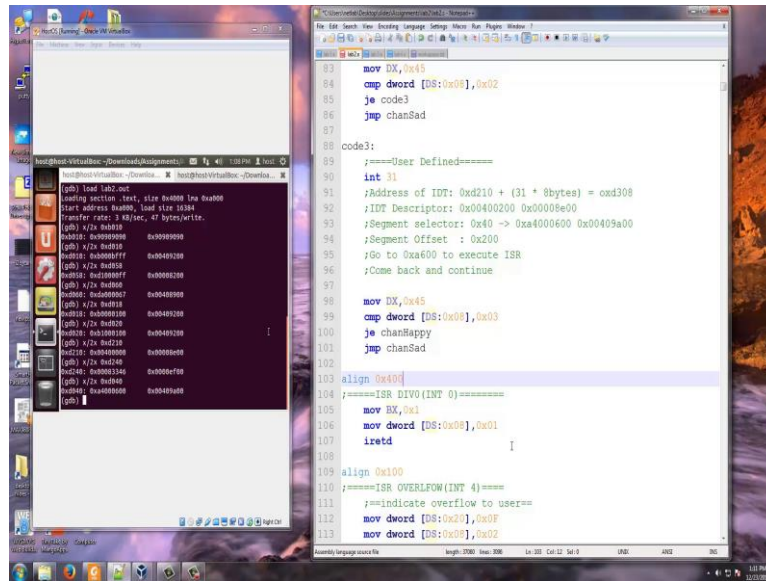


```
code2:
41  popfd
42  ;=====YOUR CODE=====
43  ;
44  ;==Important Addresses==
45  ;IDT Base address: 0xd210
46  ;GDT Base address: 0xd000
47  mov  AX,0x10
48  mov  DS,AX ;ds starts from 0xb000
49  mov  dword [DS:0x00],0x00
50  mov  AX,0x20
51  mov  SS,AX ;ss starts from 0xb100 (Expand up, R/W)
52  mov  ESP,0x0
53  mov  EBP,0x0
54  mov  dword [DS:0x00],0xcad
55
56 Code1:
57 ;==Divide by Zero (IDT0)=====
58 mov  EAX,0x23
59 mov  BL,0x0
60 div  BL
61 ;Address of IDT: 0xd210 + 0x0 = 0xd210
62 ;IDT Descriptor: 0x00400000 0x00002e00
63 ;Segment selector: 0x40 -> 0xa4000600 0x00409a00
64 ;Segment Offset : 0x0
65 ;Go to 0xa400 to execute ISR
66 ;Come back and continue
67 cmp  dword [DS:0x00],0x01
68 je  code2
69 jmp  chanSad
70
71 code2:
```

```
hosh@hosh-VirtuaBox: ~$ ./code2.exe
Loading section .text, size 0x4000 from 0xa000
Start address 0xb000, end 0xb100
Transfer rate: 3 kB/sec, 47 bytes/write.
(gdb) x/2x 0xb000
0xb000: 0x00000000
(gdb) x/2x 0xb010
0xb010: 0x00409200
(gdb) x/2x 0xb020
0xb020: 0xf01000ff
(gdb) x/2x 0xb030
0xb030: 0xd0000007
(gdb) x/2x 0xb040
0xb040: 0x00000100
(gdb) x/2x 0xb050
0xb050: 0x00409200
(gdb) x/2x 0xb060
0xb060: 0x00409200
(gdb) x/2x 0xb070
0xb070: 0x00000000
(gdb) x/2x 0xb080
0xb080: 0x00000000
(gdb) x/2x 0xb090
0xb090: 0x00000000
(gdb) x/2x 0xb0a0
0xb0a0: 0x00000000
(gdb) x/2x 0xb0b0
0xb0b0: 0x00000000
(gdb) x/2x 0xb0c0
0xb0c0: 0x00000000
(gdb) x/2x 0xb0d0
0xb0d0: 0x00000000
(gdb)
```

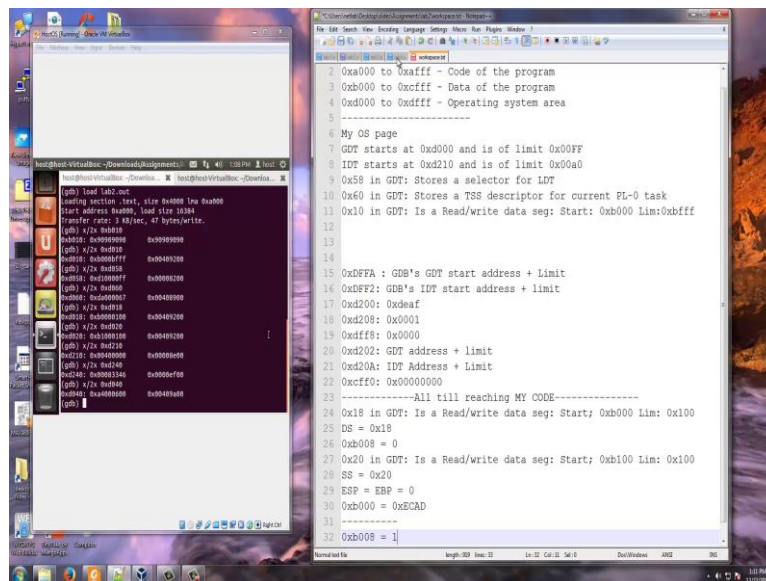
Now, I am moving EAX to 0x23 PL is 0 and say divide PL, this division by 0. So now what happens, immediately division by 0 is what? Division by 0 is interrupt number 0. We need to go and execute the first vector in the IDT, where IDT is stored? IDT is stored in 0xd210. Now, let us go and see what is there in the first descriptor, x slash 2x 0xd210. So this gives us that you need to take this segment selector 40 and do an offset of 0 and execute that code. What is there is segment selector 40? X slash 2x0xd240 please note that no d040 slash 2xd0. This starts at a400 and is of size 600 and it is an executable segment of privilege 0. So what is there in a400.

(Refer Slide Time: 11:11)



Please note that this is an align 0x400 in a000, just because this code is loaded a000. So ultimately it will come to a400. In a000 what we do? We make BX as 1 and we mov d word DS colon 0x08 0x01. What is DS? DS is b000, so b008 now becomes 1.

(Refer Slide Time: 11:38)

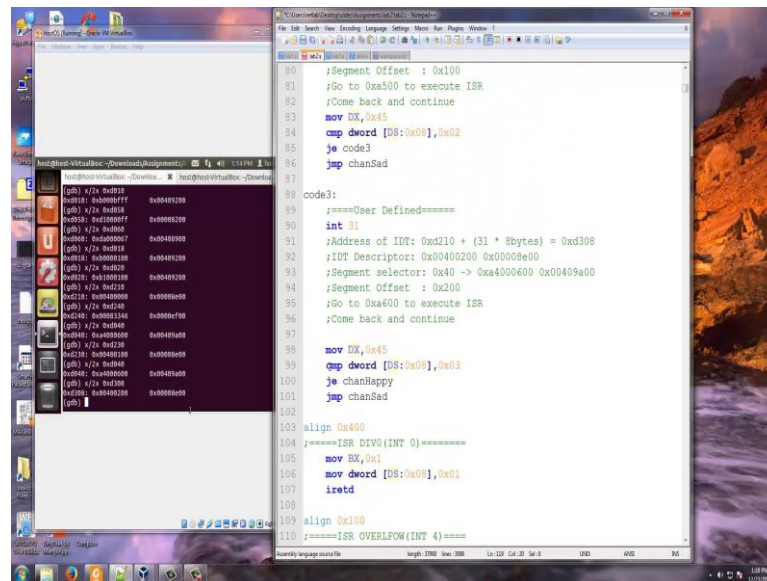


Now, b008 actually becomes 1 and you do an iret d, iret d will basically this being at interrupt it will come back to this dev BL. Now BL is 1, so EAX actually becomes 23. Now what is DS? DS is b000, so again to b008 already there is 1 so I compare b008 with 1 and if it is 1 I jmp to code 2 otherwise I jmp to chanSad which is an error. Please note that DS colon 0x08 which is basically b008 is made 1, it is actually 0 here please note here that this is 0 here it was make 0 and this became 1 only by the interrupt service routine which starts at align 0 is 400. This is the person who made it 1 and now if I get a 1 here, please note that if I get a 1 here that means the interrupt has been taken and you come back here.

Now, in code 2 what we do is we just move EAX with the large value and move EBX and we add EAX and EBX ultimately there will be overflow. Now I do into which means if there is an overflow do an interrupt that was it means and this is 4. So, 4 mean that it is now in d230. What is the interrupt vector x slash 2x0xd230, because it is starts at d210 plus so this is interrupt 4 would be d210.

It essentially says that take the segment descriptor at 0x40. So, what is the segment descriptor at 0x40? d040. This a400 and the segment offset is 100. And please note that here the segment of set is 100, in d230 a segment offset is already 100. So, I start with a400 presented, go and execute the code at a500 that what is what it says. Already I put align 400 here, so this would have align to a400. Now this align 0x centered will bring to a500. So what this does is, it moves DS colon 20 x0xf and mov dword DS colon 08 as 2. Essentially, your b008 now becomes 2, it was initially 1 now it became 2 and your another location here which is 0xb020 becomes 0x0f just to say.

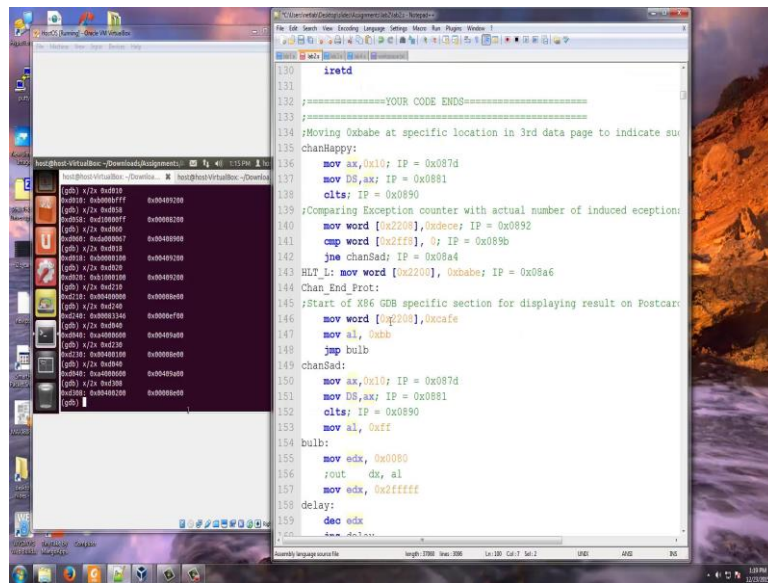
(Refer Slide Time: 15:46)



The screenshot shows a Windows desktop with a debugger window open. The debugger window is split into two panes. The left pane displays a memory dump with addresses and hex values. The right pane displays assembly code with comments. The assembly code includes instructions like `mov DX, 0x45`, `cmp dword [DS:0x08], 0x02`, `je code3`, and `jmp chanSad`. The memory dump shows values like `0x00000000`, `0x00000001`, `0x00000002`, etc.

Then it does an `iret d`, please note that when it does an `iret d` this basically comes to the next instruction after `into` unlike the previous `divide by 0` where written the same instruction this into being an interrupt it comes a next instruction. Now, what we do `mov DX, 0x45`, we just see what is `DX 0x45` x slash `2x0xb04` `mov DX, 0x45`. This is basically `DS`. So this is just a debug thing. Compare `dword [DS:0x08]` with `02`. So, `b008` now is compared with `02`, nobody would have made it `02` except the interrupt services routine because initially it was `01` here now I am comparing it with `d02`. And if it is equal I go to `code3` otherwise I `jmp chanSad`. `code3` is just interrupting `31`, so we need to go to the interrupt vector `d308` because `31` is `31` into `8` bytes and it was `d308`. So, `x slash 0x2x d308` and there you see that this is again `40`, but with offset `200`. So, `40` is again as we have seen `0x40` is code segment starting at `a400` and now with an offset `200` it is `600`. Please note that this is already `a500`, so this will be `a600`. So we go and `mov` to `DS colon 0xb030` the value `DEF`. We also make `0xb008` as `3`. And it does an `iret d`, again it basically comes to post interrupt `31`. Now `mov DX, 0x45`, compare `dword [DS:0x08]` with `3`, who would have made it `3`? Only the interrupt service routine could have made it `3`. If it is equal I `jmp` to `happy` otherwise I am sad. In `happy` again I go to `mov ax, 10`.

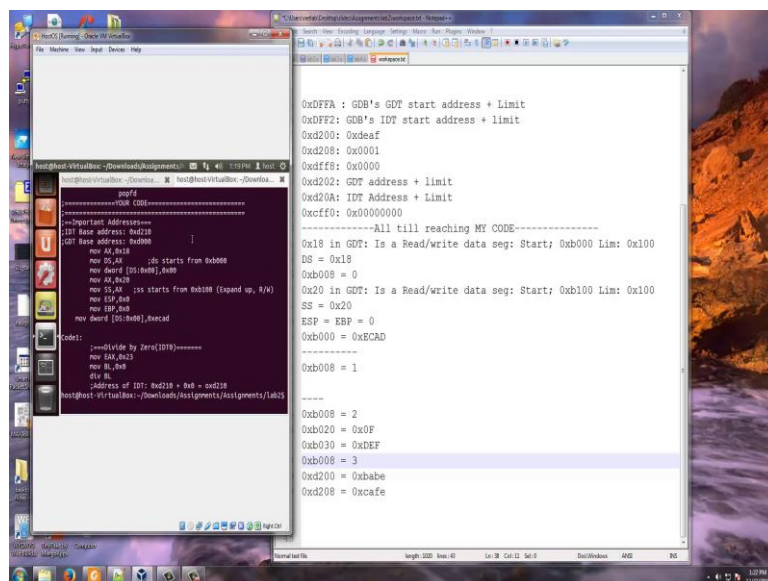
(Refer Slide Time: 18:51)



The screenshot shows a debugger interface with two main windows. The left window displays a memory dump with addresses and hex values. The right window shows assembly code with comments. The assembly code includes instructions like `iretd`, `mov ax, 0x10`, `mov DS, ax`, `cmp word [0x2200], 0x0000`, `jne chanSad`, `HLT 1: mov word [0x2200], 0xbabe`, `Chan_End_Proc`, `mov word [0x2200], 0xcafe`, `mov al, 0xb`, `jmp bulb`, `chanSad: mov ax, 0x10`, `mov DS, ax`, `mov al, 0xff`, `bulb: mov edx, 0x0000`, `out dx, al`, `mov edx, 0x2ffff`, `delay: dec edx`.

So, ultimately I need to have 2200 as (Refer Time: 18:56) what is 2200? So b plus c plus d so 0xd200 it will be 0x (Refer Time: 19:05) and then d200 and d208 will be cafe. And that ends this code. They load back the original GDT and IDT and they jmp to 0x8 fin and they do the end 3 (Refer Time: 19:40). So, this is ultimately what happens in this entire code. Now, we will just do continue right and it work now.

(Refer Slide Time: 19:55)



The screenshot shows a debugger interface with two main windows. The left window displays assembly code with comments. The right window shows memory dump with addresses and hex values. The assembly code includes comments like `;;Import Address==`, `;;Base address: 0x2210`, `;;GDT Base address: 0xb000`, `mov AX, 0x10`, `mov DS, AX`, `mov dsord [0:0x00], 0x00`, `mov SS, 0x20`, `mov ESP, 0x0`, `mov EBX, 0xb`, `;;Code: jmp 0x10 by 0x0 [0x10]=====`, `mov EBX, 0xb`, `mov B, 0xb`, `;;Address of IDT: 0xd210 = 0xb + 0xd210`. The memory dump shows addresses from 0xd0ff0 to 0xd208 and their corresponding hex values.

Let us go and see what is the value of b008, sorry b080 sorry, where is DS, is planning to a000, no-no, I think it is not loaded properly Lab2 dot out.