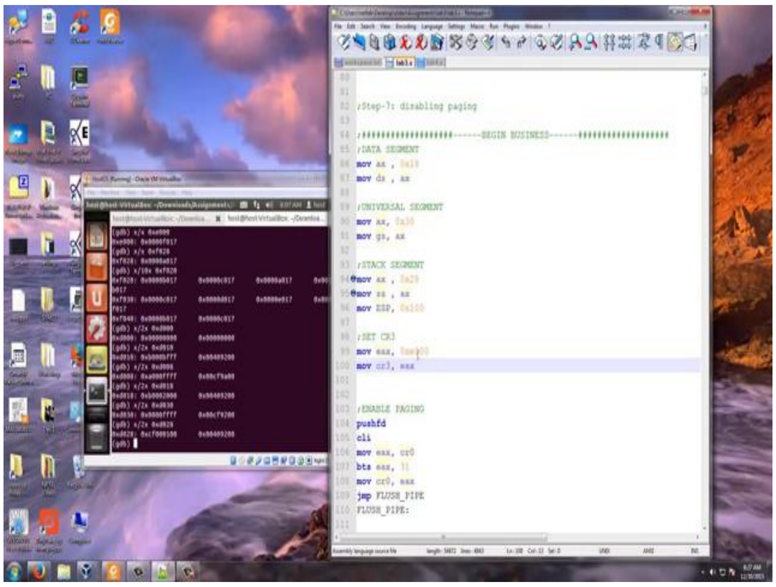
Information Security – II

Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 29
Lab3 Part 2 - Week 5

(Refer Slide Time: 00:09)

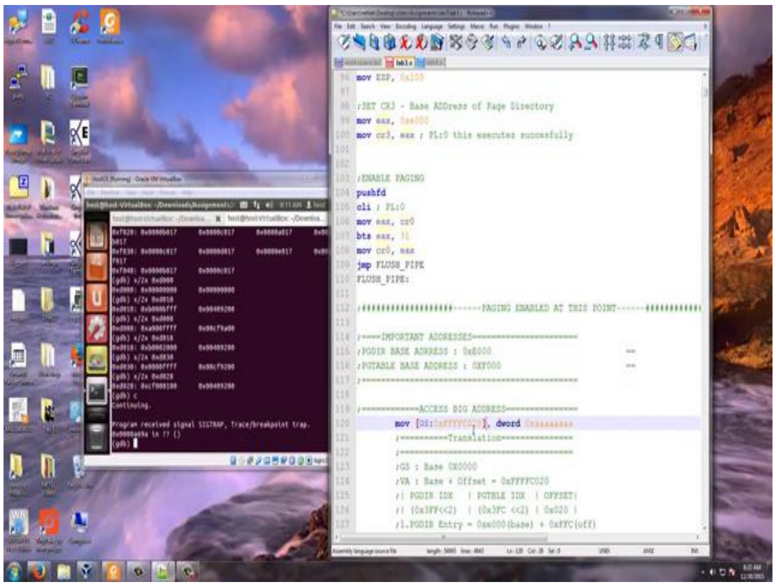


```
(gdb) x/x 0xe000
0xe000: 0x0000f017
(gdb) x/x 0xf020
0xf020: 0x0000a017
(gdb) x/10x 0xf020
0xf020: 0x0000b017      0x0000c017      0x0000a017      0x00
b017
0xf030: 0x0000c017      0x0000d017      0x0000e017      0x00
f017
0xf040: 0x0000b017      0x0000c017
(gdb) x/2x 0xd000
0xd000: 0x00000000      0x00000000
(gdb) x/2x 0xd010
0xd010: 0x0000bfff      0x00409200
(gdb) x/2x 0xd008
0xd008: 0xa000ffff      0x00cf9a00
(gdb) x/2x 0xd018
0xd018: 0x00002000      0x00409200
(gdb) x/2x 0xd030
0xd030: 0x0000ffff      0x00cf9200
(gdb) x/2x 0xd028
0xd028: 0xcf000100      0x00409200
(gdb)
```

```
80
81
82  ;Step-7: disabling paging
83
84  ;###################------BEGIN BUSINESS------###################
85  ;DATA SEGMENT
86  mov ax , 0x10
87  mov ds , ax
88
89  ;UNIVERSAL SEGMENT
90  mov ax, 0x30
91  mov gs, ax
92
93  ;STACK SEGMENT
94  mov ax , 0x28
95  mov ss , ax
96  mov ESP, 0x100
97
98  ;SET CR3
99  mov eax, 0xe000
100 mov cr3, eax
101
102
103 ;ENABLE PAGING
104 pushfd
105 cli
106 mov eax, cr0
107 bts eax, 31
108 mov cr0, eax
109 jmp FLUSH_PIPE
110 FLUSH_PIPE:
111
```

Now, what I am doing here in line number 99, see I am pushing e000 and I am moving e000 to cr3. Note that, this is actually a privilege instruction, since the PL is 0. This executes successfully because currently the code segment has privilege zero and it is set in line number, you know here, already when we enter the code, so this instruction will be executed correctly.

(Refer Slide Time: 00:53)



```
0xf020: 0x0000b017      0x0000c017      0x0000a017      0x00
b017
0xf030: 0x0000c017      0x0000d017      0x0000e017      0x00
f017
0xf040: 0x0000b017      0x0000c017
(gdb) x/2x 0xd000
0xd000: 0x00000000      0x00000000
(gdb) x/2x 0xd010
0xd010: 0xb0000fff      0x00409200
(gdb) x/2x 0xd008
0xd008: 0xa000ffff      0x00cf9a00
(gdb) x/2x 0xd018
0xd018: 0xb0002000      0x00409200
(gdb) x/2x 0xd030
0xd030: 0x0000ffff      0x00cf9200
(gdb) x/2x 0xd028
0xd028: 0xcf000100      0x00409200
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000a09a in ?? ()
(gdb)
```

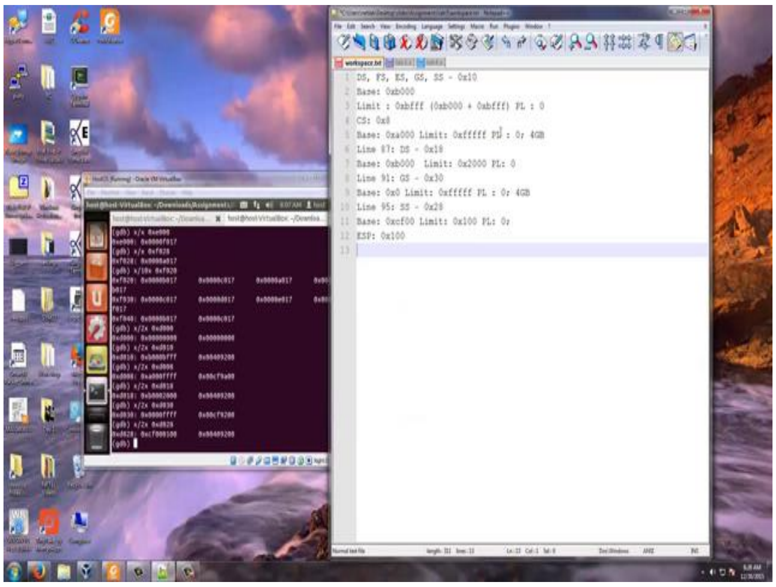
```
mov ESP, 0x100

;SET CR3 - Base ADDress of Page Directory
mov eax, 0xe000
mov cr3, eax ; PL:0 this executes succesfully


;ENABLE PAGING
pushfd
cli ; PL:0
mov eax, cr0
bts eax, 31
mov cr0, eax
jmp FLUSH_PIPE
FLUSH_PIPE:

;###################------PAGING ENABLED AT THIS POINT------###########

;====IMPORTANT ADDRESSES====================
;PGDIR BASE ADRRESS : 0xE000                              ==
;PGTABLE BASE ADDRESS : 0XF000                            ==
;===========================================

;===========ACCESS BIG ADDRESS==============
        mov [GS:0xFFFFC020], dword 0xaaaaaaaa
        ;==========Translation==============
        ;==================================
        ;GS : Base 0X0000
        ;VA : Base + Offset = 0xFFFFC020
        ;| PGDIR IDX   | PGTBLE IDX  | OFFSET|
        ;| (0x3FF<<2)  | (0x3FC <<2) | 0x020 |
        ;1.PGDIR Entry = 0xe000(base) + 0xFFC(off)
```

So, what I have done now into cr3, I am moving the base address of my page directory. This is having the base address of page directory. Now, I am enabling paging. So, I am pushing the flag and it will work correctly because the stack is well within the limit. Now, I am clearing the interrupt again, this will be successful again because of my PL 0 code. When the paging is enabled, at that point of time when interrupt comes it actually sought of becomes like an imprecise exception because we do not know whether the paging will happen there. Now, the interrupt service routine, where it will go? Where it will start executing? Because you are in the middle of a paging, so at that point of time I do not want any interrupt to happen when the paging is getting enabled because we really do not know the state of the operating system, the state of the architecture.

We clear the interrupt flag. So, here we store the state of the flags in the stack and then I go and clear it, clear the inter flag in the flag register. Now, I am moving the content of cr0 into eax, I cannot directly go and operate on eax, I could only move values from there. Then I am going, I am setting the thirty first bit and then I am moving the content of eax back to cr0. At this point of time completely my paging is now enabled. The next instruction to be executed will be this jump instruction because my a000 page is identically mapped. If my a000 page was not identically mapped then the next instruction to be executed will be as per the translation. So, it will go to some other page since my a000 is mapped down to a000. This will execute this and I do an unconditional jump here because many of the architectures are out of order architectures. So, there can be some instruction that is after this could be still in execution would have started execution. There will be some instruction before which is still to get completed and we do not know which status it is. So, we would like to completely disable those instructions at least the one that is happening after this. That is very, very important because those instruction have to now execute with paging enabled and if they had progressed to some level and then what would have happened there is that, they will start executing as if paging is not enabled and that can basically lead to some program incorrectness.

It is very important that we first enter pipeline specifically in out of order execution processes to see that the execution is actually correct. Once this is all done, you now have a page directory in 0xe000 and base table as 0xf000 as we demonstrated earlier. Now, you see what I am going to do, I am doing start executive is program. So, here I am going to make see, it has executed here on the left hand side.

(Refer Slide Time: 04:46)

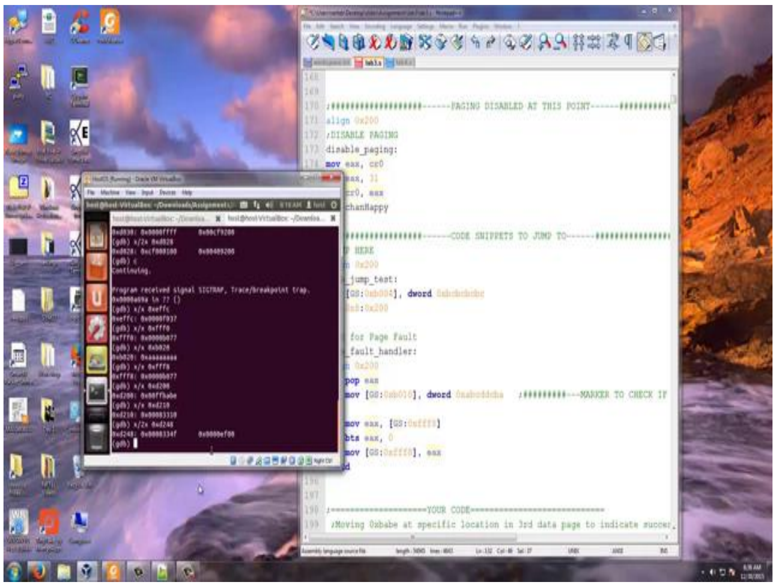


```
(gdb) x/x 0xe000
0xe000: 0x0000f017
(gdb) x/x 0xf028
0xf028: 0x0000a017
(gdb) x/10x 0xf020
0xf020: 0x0000b017      0x0000c017      0x0000a017
b017
0xf030: 0x0000c017      0x0000d017      0x0000e017
f017
0xf040: 0x0000b017      0x0000c017
(gdb) x/2x 0xd000
0xd000: 0x00000000      0x00000000
(gdb) x/2x 0xd010
0xd010: 0xb000ffff      0x00409200
(gdb) x/2x 0xd008
0xd008: 0xa000ffff      0x00cf9a00
(gdb) x/2x 0xd018
0xd018: 0xb0002000      0x00409200
(gdb) x/2x 0xd030
0xd030: 0x0000ffff      0x00cf9200
(gdb) x/2x 0xd028
0xd028: 0xcf000100      0x00409200
(gdb)
```

```
DS, FS, ES, GS, SS - 0x10
Base: 0xb000
Limit : 0xbfff (0xb000 + 0xbfff) PL : 0
CS: 0x8
Base: 0xa000 Limit: 0xfffff PL : 0, 4GB
Line 87: DS - 0x18
Base: 0xb000  Limit: 0x2000 PL: 0
Line 91: GS - 0x30
Base: 0x0 Limit: 0xfffff PL : 0, 4GB
Line 95: SS - 0x28
Base: 0xcf00 Limit: 0x100 PL: 0,
ESP: 0x100
```

What execution happens? I am doing gs code and ffff0c020. Note, what is gs? Please note that, gs has 0 as base and very large limit 5 years. So, this will go to ffffc020 and I moving the word aaaa, but ffffc020 may not exist in your system. So, this ffff0c020 essentially has to get translated. How it gets translated? Please note that, your page directory index would be the last ten bits, which is 3fff2 and your base table would be the next ten bits which is 3fc and right shift because each entry is 4 bytes and then the offset is 20.

(Refer Slide Time: 05:36)



```
0xd030: 0x0000ffff      0x00cf9200
(gdb) x/2x 0xd028
0xd028: 0xcf000100      0x00409200
(gdb) c
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000a69a in ?? ()
(gdb) x/x 0xeffc
0xeffc: 0x0000f037
(gdb) x/x 0xfff0
0xfff0: 0x0000b077
(gdb) x/x 0xb020
0xb020: 0xaaaaaaaa
(gdb) x/x 0xfff8
0xfff8: 0x0000b077
(gdb) x/x 0xd200
0xd200: 0x00ffbabe
(gdb) x/x 0xd210
0xd210: 0x00003310
(gdb) x/2x 0xd248
0xd248: 0x0000334f      0x0000ef00
(gdb)
```

```
;###################------PAGING DISABLED AT THIS POINT------##########
align 0x200
;DISABLE PAGING
disable_paging:
mov eax, cr0
    eax, 31
    cr0, eax
    chanHappy

    ################------CODE SNIPPETS TO JUMP TO------################
    P HERE
    n 0x200
    _jump_test:
    [GS:0xb004], dword 0xbcbcbcbc
    0x8:0x200

    for Page Fault
    _fault_handler:
    : 0x200
    pop eax
    mov [GS:0xb010], dword 0xabcddcba   ;#########---MARKER TO CHECK IF

    mov eax, [GS:0xfff8]
    bts eax, 0
    mov [GS:0xfff8], eax
    d


;===================YOUR CODE============================
 ;Moving 0xbabe at specific location in 3rd data page to indicate succes
```

So, now we have to go and look at 0xeffcx slash x0xeffc. Please note that, this is pointing to f000 and now you see that this is become 3. If you have noticed earlier it was 1, now why it became 3 because the access bit has become 1, the architecture because you have accessed this page, this particular instruction has accessed this address, this entry in the page directory and that is why we see that this is f037 and when you study operating systems there you will have some replacement strategies like least recently used, etcetera.

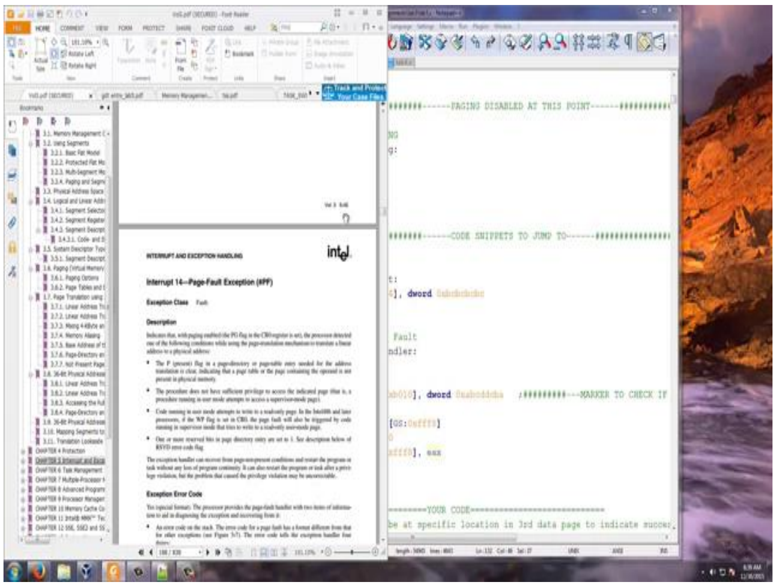
This access bit can essentially give you some indication of which are the pages that were accessed at the recent past because the program is executing after the program executes is probably generates the page fault. Now, the page fault handler has to basically start executing other point. It should know which page has been accessed in the recent. So, this access bit can give information to the operating system, especially the interrupt service routine of the operating system, which is the page fault handler to basically tell, these are the pages that have been accessed. So, this access bit is very important. Now, from this I go to the f000 page and there now what is the page table entry there, which is fff0 because 3fc right shift 2 bits will give me F of 0. So, I will go and see x slash x0xfff0 and note that, this is again mapped down to the b000 page.

What this particular instruction in line number 120 should have done? It should have made v020 as aaaaa, yes, it has made v020 as aaaaa. So, this is a very huge address but true page translation I have got and made b020 as this map on to b020. So, this basically explains you, how this in architectures they call it as page walk. How the page walk actually happens in the xxx architecture.

Now, let us go and see, I want to generate a page fault and I already told you that the page corresponding to fff8, I will repeat this is not available. So, that we actually saw it now the page fault handler has handled this. Now, let us go and see when we did in the previous thing what has happened? We saw that this F of F8 page was not present. Then what happens when this particular command executes, immediately the page fault is generated. The page fault number is ee and your interrupt service routine is at 0x, interrupt descriptor table is set e200. Your IDT is at d210.

So, now we are now looking at x slash2x0xd21e right d210141 would be, it should be 14 into 4 that is 38. So, 28 would be the entry. So, d248 and this essentially says that d248 is the entry and the interrupt service routine is at this is the code segment 0008, this is the code segment and within that code segment I need to go through an offset of 334f. Now, where is the code segment, what is 0008 code segment, it is a000 and from there I have to go to an index of 334f. So, so d34f would be the interruption service routine. Now, I think we have to do a little more calculation; the page fault. Let us just go and see, what is the page fault number?

(Refer Slide Time: 11:14)



INTERRUPT AND EXCEPTION HANDLING

Interrupt 14—Page-Fault Exception (#PF)

Exception Class Fault

Description

[illegible]

Exception Error Code

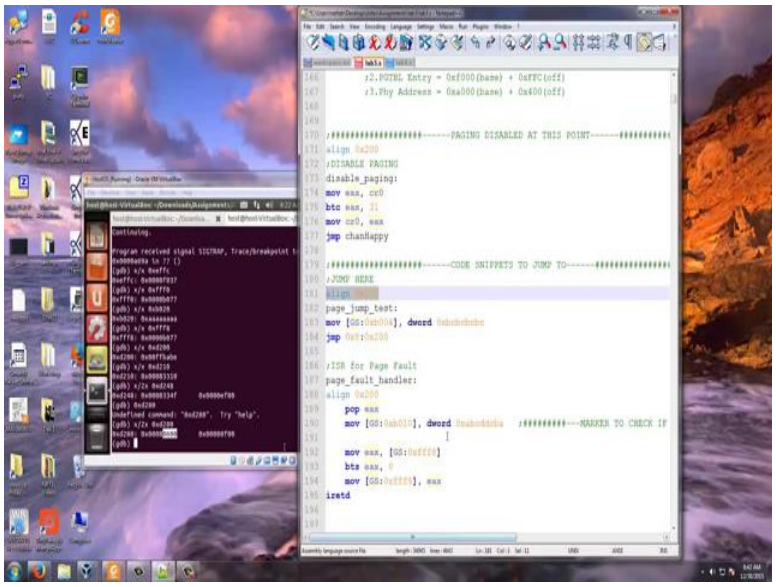
[illegible]

```
#######------PAGING DISABLED AT THIS POINT------##########
NG
g:

#######------CODE SNIPPETS TO JUMP TO------###############

t:
], dword [illegible]

 Fault
ndler:

[illegible]], dword [illegible]   ;#########---MARKER TO CHECK IF
[GS:[illegible]]

[illegible]], eax

========YOUR CODE=============================
be at specific location in 3rd data page to indicate succe
```

So, let us undergo, this interrupt number 11 is segment not present, interrupt number 12 is tag fault, interrupt 13 is general protection fault, interrupt 14 is page fault. Now, 14 is actually e. So, 14 into 4 bytes each that will give us 56 and 56 is 38 in hexadecimal and your inter descriptor tables starts at d210, d210 plus 38, it should be d248 only and so your interrupt service routine is at d3f4. Sorry, now where are we going wrong, I am just making some mistakes, so that you can also realize what the mistake is each of this entry?

How many bytes it is 8 bytes, so your inter descriptor table, it will be your 112th byte not the 56th byte. So, what is 112 in hexadecimal, it will be 70. So, I have to look at 0xd280, right. Now, you see that, you have to go to the code segment 8 and go for an offset of 600. So, what is code segment 8 here? It is code segment is 0x8, it has base at a000 and 600 offset would be a000 600. Now, this a000 600 when it gets translated will again be a000 600 because it is identically mapped. This is also one of the reason, why the operating system page should be identically mapped otherwise you land up in bigger problem. So, what is a000 600. Please note that, when I do this align to I started a000 and I have done.

(Refer Slide Time: 14:58)



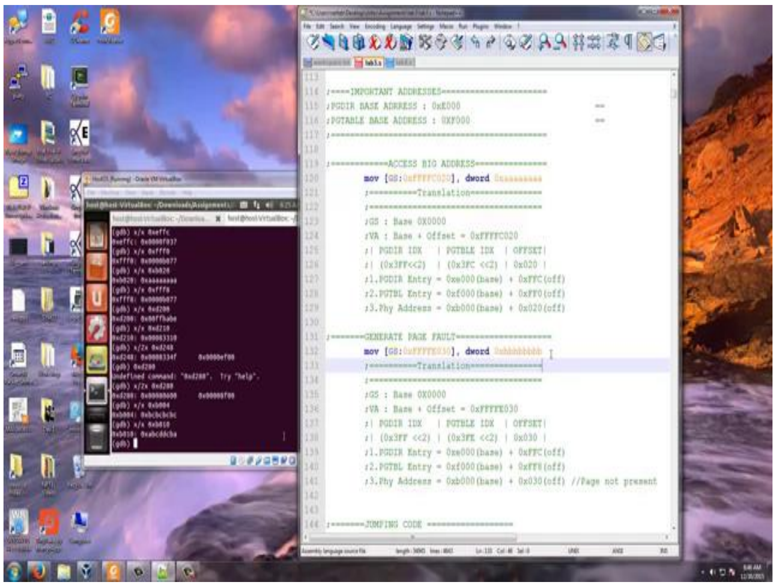
```
continuing.

Program received signal SIGTRAP, Trace/breakpoint t
0x0000a00a in ?? ()
(gdb) x/x 0xeffc
0xeffc: 0x0000f037
(gdb) x/x 0xfff8
0xfff8: 0x0000b077
(gdb) x/x 0xb020
0xb020: 0xaaaaaaaa
(gdb) x/x 0xfff8
0xfff8: 0x0000b077
(gdb) x/x 0xd200
0xd200: 0x00ffbabe
(gdb) x/x 0xd210
0xd210: 0x00003310
(gdb) x/2x 0xd240
0xd240: 0x0000334f      0x0000ef00
(gdb) 0xd200
Undefined command: "0xd200".  Try "help".
(gdb) x/2x 0xd200
0xd200: 0x00000000      0x0000ef00
(gdb)
```

```
        ;2.PGTBL Entry = 0xf000(base) + 0xFFC(off)
        ;3.Phy Address = 0xa000(base) + 0x400(off)


;###################------PAGING DISABLED AT THIS POINT------##########
align 0x200
;DISABLE PAGING
disable_paging:
mov eax, cr0
btc eax, 31
mov cr0, eax
jmp chanHappy

;###################------CODE SNIPPETS TO JUMP TO------###############
;JUMP HERE
align 0x200
page_jump_test:
mov [GS:0xb004], dword 0xbcbcbcbc
jmp 0x8:0x200

;ISR for Page Fault
page_fault_handler:
align 0x200
    pop eax
    mov [GS:0xb010], dword 0xabcddcba    ;#########---MARKER TO CHECK IF

    mov eax, [GS:0xfff8]
    bts eax, 0
    mov [GS:0xfff8], eax
iretd
```

I have done one align, align 200 here, already it would have crossed 200 there. So, this would have become, a000 400 and I do another align is 0x200. This align a000 600, your base fault will start happening here. This is your inter service routine for your page fault and what we do here, we just do some steps. We just put g s colon 0xb0004 as this, essentially, to prove that I have entered the page fault handler. So, what is g s colon 0xb0004, this is basically bcbcbcbc. So, this again it will map on to 0xb0004 because the b000 page is identically mapped and there you see the set of bcbcbcbc, right. So, it is set here and then I just go and do a jump 0x8 colon 0x200 here. So, this is align 0x200, this is align 400 and this is align 0x200, sorry. So, your page fault handler actually starts executing here. It pops eax. It pops a value from the stack that will be the return address here. So, that is eax then now it moves g s colon v010 as abcddcba.

So, let us see x slash x0xb010 and that is adabcddcba and then. So, it had entered the handler now to this eax. So, what I am doing here is, I am moving the content offff8 to eax, what I mean by moving a content offff8 because fff8 is the base table entry in which we have marked the page as not present and I go and test and set the 0th bit because that is basically telling whether the page is present or absent. You know that initially it was b016, now it became b017 and I move the thing back to 0xfff8. So, this pop eax essentially pops out the error code and then it basically goes to the page fault handler its start executing it makes b010 as adabcddcba and then it goes and takes that particular base table entry in afff8, it makes that last bit 1, essentially, now the page will be present it moves it back to eax and then what it does is, it returns when it does a I read d we come back to the same instruction right the same instruction which is which is here.

(Refer Slide Time: 18:54)

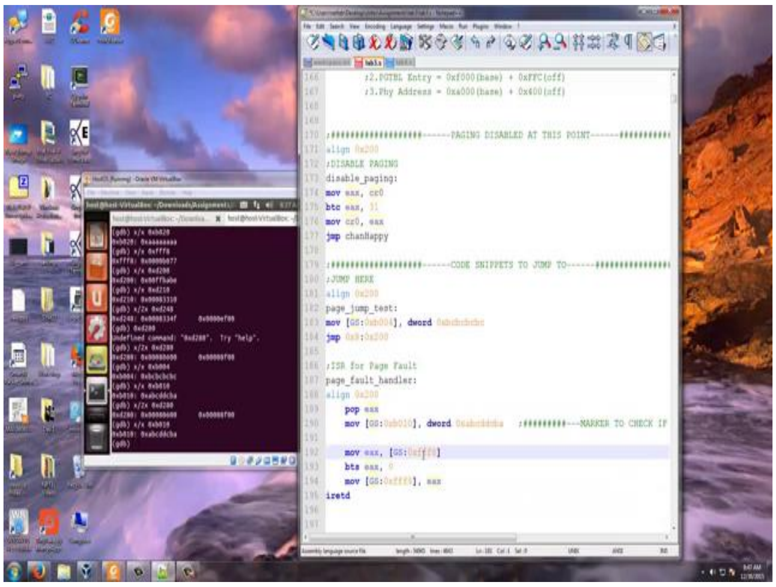


```
(gdb) x/x 0xeffc
0xeffc: 0x0000f037
(gdb) x/x 0xfff0
0xfff0: 0x0000b077
(gdb) x/x 0xb020
0xb020: 0xaaaaaaaa
(gdb) x/x 0xfff8
0xfff8: 0x0000b077
(gdb) x/x 0xd200
0xd200: 0x00ffbabe
(gdb) x/x 0xd210
0xd210: 0x00083310
(gdb) x/2x 0xd240
0xd240: 0x0008334f      0x0000ef00
(gdb) 0xd280
Undefined command: "0xd280".  Try "help".
(gdb) x/2x 0xd280
0xd280: 0x00080600      0x00008f00
(gdb) x/x 0xb004
0xb004: 0xbcbcbcbc
(gdb) x/x 0xb010
0xb010: 0xabcddcba
(gdb)
```

```
;====IMPORTANT ADDRESSES=====================
;PGDIR BASE ADRRESS : 0xE000
;PGTABLE BASE ADDRESS : 0XF000
;============================================


;============ACCESS BIG ADDRESS===============
        mov [GS:0xFFFFC020], dword 0xaaaaaaaa
        ;=========Translation===============
        ;==================================
        ;GS : Base 0X0000
        ;VA : Base + Offset = 0xFFFFC020
        ;| PGDIR IDX   | PGTBLE IDX  | OFFSET|
        ;| (0x3FF<<2)  | (0x3FC <<2) | 0x020 |
        ;1.PGDIR Entry = 0xe000(base) + 0xFFC(off)
        ;2.PGTBL Entry = 0xf000(base) + 0xFF0(off)
        ;3.Phy Address = 0xb000(base) + 0x020(off)

;=======GENERATE PAGE FAULT===================
        mov [GS:0xFFFFE030], dword 0xbbbbbbbb
        ;=========Translation===============
        ;==================================
        ;GS : Base 0X0000
        ;VA : Base + Offset = 0xFFFFE030
        ;| PGDIR IDX   | PGTBLE IDX  | OFFSET|
        ;| (0x3FF <<2) | (0x3FE <<2) | 0x030 |
        ;1.PGDIR Entry = 0xe000(base) + 0xFFC(off)
        ;2.PGTBL Entry = 0xf000(base) + 0xFF8(off)
        ;3.Phy Address = 0xb000(base) + 0x030(off) //Page not present


;=======JUMPING CODE =================
```

Now, I go and execute this instruction. This instruction, now will be successfully executed because the page fault is now handled, the page now has become present. So, let me very quickly summarize, what we have done, we knew that the page corresponding to this was not marked, not present in the page table. I go and access this this page and I want to write into this page. This eventually creates a page fault, page fault is interrupt number 14, which is going to be in the 112th byte in your page fault handler. So, I go and look at the 112th byte, 112 is 70. So, d210 is the first byte of interrupt handler interrupt descriptor table. So, I go and look at the entry at d280 and there I actually, I find that I am having the code descriptor 8 and I am having 600 as the base.

(Refer Slide Time: 20:03)

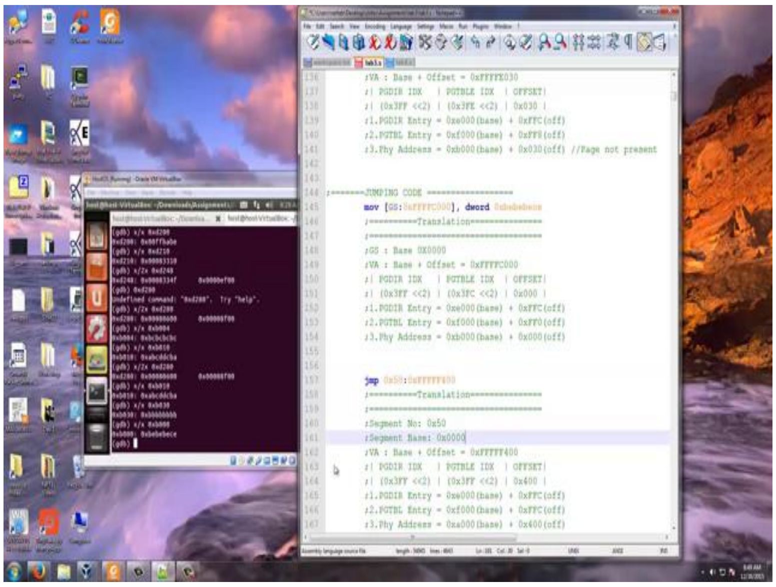


```
(gdb) x/x 0xb020
0xb020: 0xaaaaaaaa
(gdb) x/x 0xfff8
0xfff8: 0x00000b077
(gdb) x/x 0xd200
0xd200: 0x00ff5abe
(gdb) x/x 0xd210
0xd210: 0x00003310
(gdb) x/2x 0xd248
0xd248: 0x0000334f      0x0000ef00
(gdb) 0xd200
Undefined command: "0xd200".  Try "help".
(gdb) x/2x 0xd200
0xd200: 0x00000000      0x00008f00
(gdb) x/x 0xb004
0xb004: 0xbcbcbcbc
(gdb) x/x 0xb010
0xb010: 0xabcddcba
(gdb) x/2x 0xd200
0xd200: 0x00000600      0x00008f00
(gdb) x/x 0xb010
0xb010: 0xabcddcba
(gdb)
```

```
166         ;2.PDTBL Entry = 0xf000(base) + 0xFFC(off)
167         ;3.Phy Address = 0xa000(base) + 0x400(off)
168
169
170 ;###################------PAGING DISABLED AT THIS POINT------##########
171 align 0x200
172 ;DISABLE PAGING
173 disable_paging:
174 mov eax, cr0
175 btc eax, 31
176 mov cr0, eax
177 jmp chanHappy
178
179 ;###################------CODE SNIPPETS TO JUMP TO------###############
180 ;JUMP HERE
181 align 0x200
182 page_jump_test:
183 mov [GS:0xb004], dword 0xbcbcbcbc
184 jmp 0x8:0x200
185
186 ;ISR for Page Fault
187 page_fault_handler:
188 align 0x200
189     pop eax
190     mov [GS:0xb010], dword 0xabcddcba   ;#########---MARKER TO CHECK IF
191
192     mov eax, [GS:0xfff8]
193     bts eax, 0
194     mov [GS:0xfff8], eax
195 iretd
196
197
```

So, now with that I start executing at a000 600 please note that this is a000 200, 171 line, this is a000 400 and this is a000 600. I first pop out the error code then I just want to show the world that I have entered this interrupt service routine. So, I go and see adcbdcba, I move into b010, which we have seen here s slash x0xb010 is this and then we move ffff8, this is the page table entry I move it to ax, I make the 0th bit has one essentially making pages present.

(Refer Slide Time: 20:43)

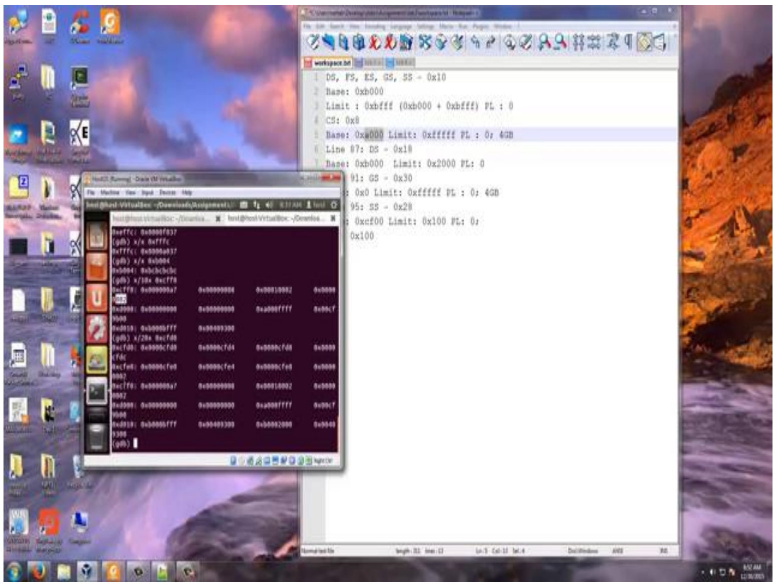


```
(gdb) x/x 0xd200
0xd200: 0x00ffbabe
(gdb) x/x 0xd210
0xd210: 0x00083318
(gdb) x/2x 0xd248
0xd248: 0x0008334f      0x0000ef00
(gdb) 0xd200
Undefined command: "0xd200".  Try "help".
(gdb) x/2x 0xd200
0xd200: 0x00000000      0x00000f00
(gdb) x/x 0xb004
0xb004: 0xbcbcbcbc
(gdb) x/x 0xb010
0xb010: 0xabcddcba
(gdb) x/2x 0xd200
0xd200: 0x00000000      0x00000f00
(gdb) x/x 0xb010
0xb010: 0xabcddcba
(gdb) x/x 0xb030
0xb030: 0x00000bbb
(gdb) x/x 0xb000
0xb000: 0xbebebece
(gdb)
```

```
        ;VA : Base + Offset = 0xFFFFE030
        ;| PGDIR IDX   | PGTBLE IDX  | OFFSET|
        ;| (0x3FF <<2) | (0x3FE <<2) | 0x030 |
        ;1.PGDIR Entry = 0xe000(base) + 0xFFC(off)
        ;2.PGTBL Entry = 0xf000(base) + 0xFF8(off)
        ;3.Phy Address = 0xb000(base) + 0x030(off) //Page not present


;=======JUMPING CODE ==================
        mov [GS:0xFFFFC000], dword 0xbebebebe
        ;==========Translation===============
        ;====================================
        ;GS : Base 0X0000
        ;VA : Base + Offset = 0xFFFFC000
        ;| PGDIR IDX   | PGTBLE IDX  | OFFSET|
        ;| (0x3FF <<2) | (0x3FC <<2) | 0x000 |
        ;1.PGDIR Entry = 0xe000(base) + 0xFFC(off)
        ;2.PGTBL Entry = 0xf000(base) + 0xFF0(off)
        ;3.Phy Address = 0xb000(base) + 0x000(off)


        jmp 0x50:0xFFFFF400
        ;==========Translation===============
        ;====================================
        ;Segment No: 0x50
        ;Segment Base: 0x0000
        ;VA : Base + Offset = 0xFFFFF400
        ;| PGDIR IDX   | PGTBLE IDX  | OFFSET|
        ;| (0x3FF <<2) | (0x3FF <<2) | 0x400 |
        ;1.PGDIR Entry = 0xe000(base) + 0xFFC(off)
        ;2.PGTBL Entry = 0xf000(base) + 0xFFC(off)
        ;3.Phy Address = 0xa000(base) + 0x400(off)
```

I move it back to the page table and I do and I read d and this I read d again comes and re executes this particular instruction and now you will see that this particular thing translates to b030 and now you will see s slash ax0xb030 has bbb, which is the next thing is that I go and I just want to show that I am coming here. I just see that bb this fffc0b0, this actually maps on to b000. So, I will just go and see s slash x0b000, this has bbbbc. So, this just again shows that I am coming to this part of the court.

(Refer Slide Time: 21:38)



```
0xeffc: 0x0000f037
(gdb) x/x 0xfffc
0xfffc: 0x0000a037
(gdb) x/x 0xb004
0xb004: 0xbcbcbcbc
(gdb) x/10x 0xcff0
0xcff0: 0x000000a7      0x00000000      0x00010002      0x0000
0002
0xd000: 0x00000000      0x00000000      0xa000ffff      0x00cf
9b00
0xd010: 0xb000bfff      0x00409300
(gdb) x/20x 0xcfd0
0xcfd0: 0x0000cfd0      0x0000cfd4      0x0000cfd8      0x0000
cfdc
0xcfe0: 0x0000cfe0      0x0000cfe4      0x0000cfe8      0x0000
0002
0xcff0: 0x000000a7      0x00000000      0x00010002      0x0000
0002
0xd000: 0x00000000      0x00000000      0xa000ffff      0x00cf
9b00
0xd010: 0xb000bfff      0x00409300      0xb0002000      0x0040
9300
(gdb)
```

```
DS, FS, ES, GS, SS - 0x10
Base: 0xb000
Limit : 0xbfff (0xb000 + 0xbfff) PL : 0
CS: 0x8
Base: 0xa000 Limit: 0xfffff PL : 0; 4GB
Line 87: DS - 0x18
Base: 0xb000  Limit: 0x2000 PL: 0
 91: GS - 0x30
: 0x0 Limit: 0xfffff PL : 0; 4GB
 95: SS - 0x28
: 0xcf00 Limit: 0x100 PL: 0;
 0x100
```

Now, I am jumping to a very large address 0x50 colon and a very high offset. So, what is there in 0x50? Let us go and see s slash x0xd050. So, I should say 2x, now this is a code segment of size 4 Gb with base as 0 and this is a large segment and I am doing fffff400. So, since I have 32 Gb, I can go there now this is fffff400 actually maps on to a000 400 because this maps on to the page directory entry which is fffc and so, let us go and see s slash x0xefffc and that is pointing to f0 and there I have s slash xaffffc and please note that this is also pointing to a000.

So, I have one more additional page which is pointing to a000. Just to demonstrate this, now this plus the last 12 bits is 400. So, I have to jump to a000 400 and note that we have done. So, this is align 200, this is align 400. So, it will jump to this and what it does it makes b000 4 as this thing. So, s slash x00xb004 and it has made as bcbcbc. So, it start executing and again it just jumps 0x80x 200 0x8 actually starts b000. So, it a000 it has to come the a000 200, which is this. So, after we finish this, it jumps back to line number 174 and there you go and disable paging and you happy to execute exceed the program.

This is how the entire thing works and where is our stack? So, stack is at, let us go and see where our stack is, it is acf100 and so, let me just see s slash s slash and its base is 0x100. So, cf100 plus 100 is ddd000. So, let me say s slash 10x0xeff0 and there you see that the stack had some activity. So, it has pushed a 7, it has pushed a code segment register, it has pushed some error code 002, etcetera. So, this is where the stack has been active, this is from this point. So, it does not touch t000 from cfff is where the stack is active and you see some stack activity there. If you just say s slash 20x0xcfd0, I am sorry, it is cfd0. So, there you see, we have just loaded some random data here but now you see from cff0 and these points. This is where the stack has been pushing data inside. So, you can now go and see the content of this stack and find out how they work.

I hope you understood this particular assignment. Please go through this again my lecture and also please be very good in your calculations. So, lot of times I have made certain voluntary errors and just to tell you, where you can also go wrong. A typical error that an assembly programmer does and he is trying to look at coding this is what I have also done here. So, those errors you should be very careful and if I do not to do that error, it does not come to you. So, I have voluntarily incorporated certain errors and I have rewind that back. So, please look at, what errors I have done and how I have rewind that back and that will also be a good learning experience.

Let us meet in the next session.