



操作系统

Operating Systems

Lec7 实验二 物理内存管理
清华大学计算机系

概 述

- x86 特权级 (privilege levels)
- x86 内存管理单元 (Memory Management Unit , MMU)

- 了解不同特权级的差别
- 了解当前 CPU 处在哪个特权级
- 了解特权级切换

X86 特权级

x86 特权级 - 简介

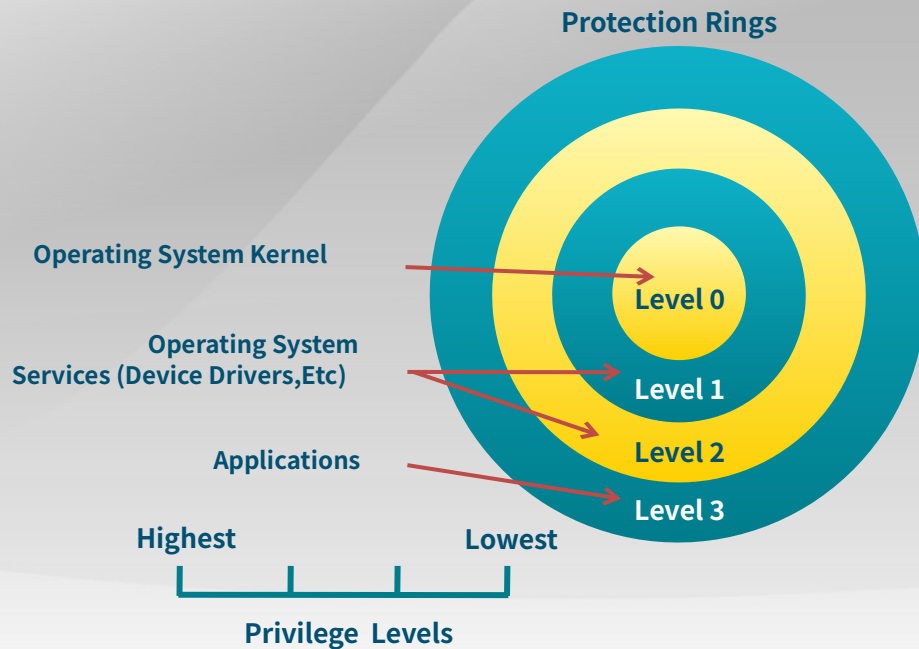


图 Protection Rings

■ Linux 和 uCore 只使用 ring 0 and ring 3

x86 特权级 - 区别 ?

- 一些指令（比如特权指令）只能执行在 ring 0 (e.g. lgdt).
- CPU 在如下时刻会检查特权级
 - ▶ 访问数据段
 - ▶ 访问页
 - ▶ 进入中断服务例程（ISRs）
 - ▶
- 如果检查失败会如何？

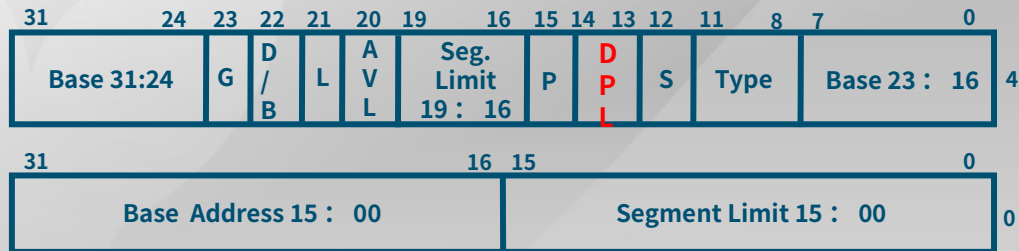
General Protection Fault!

x86 特权级 - 段选择子



段选择子 Segment Selector

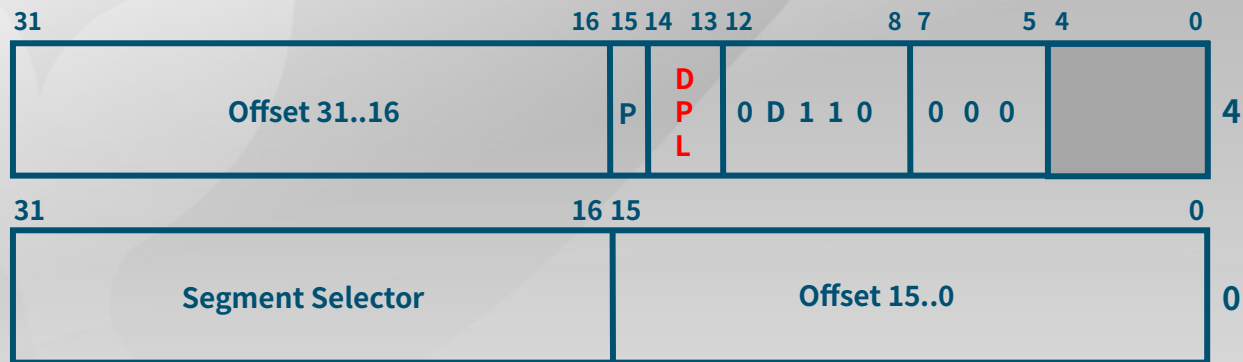
x86 特权级 - 段描述符



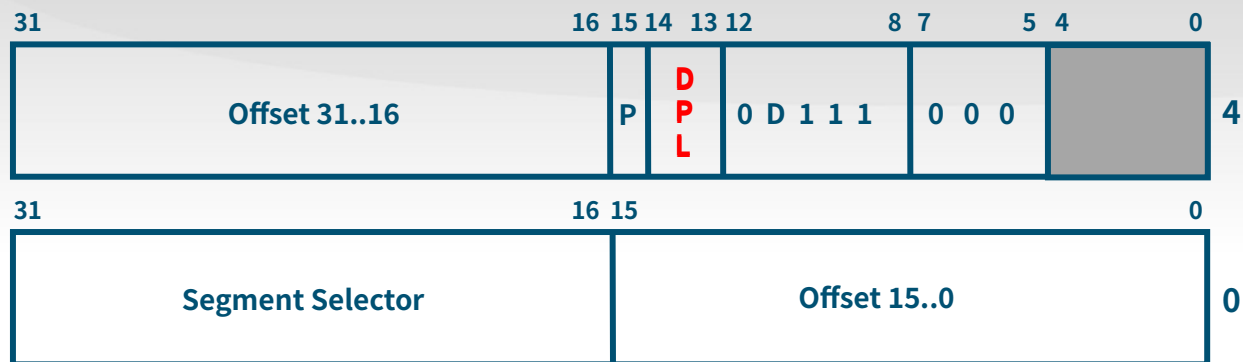
段描述符 Segment Descriptor

x86 特权级 - 门描述符

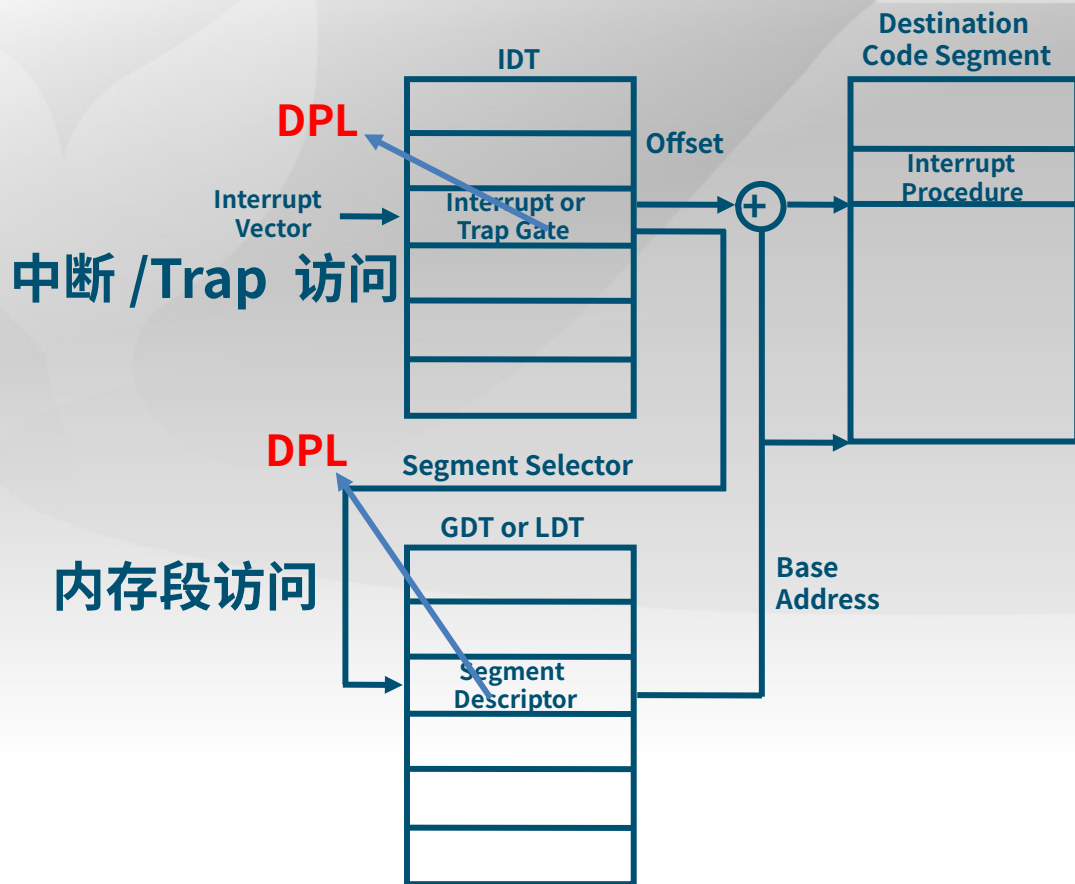
Interrupt Gate



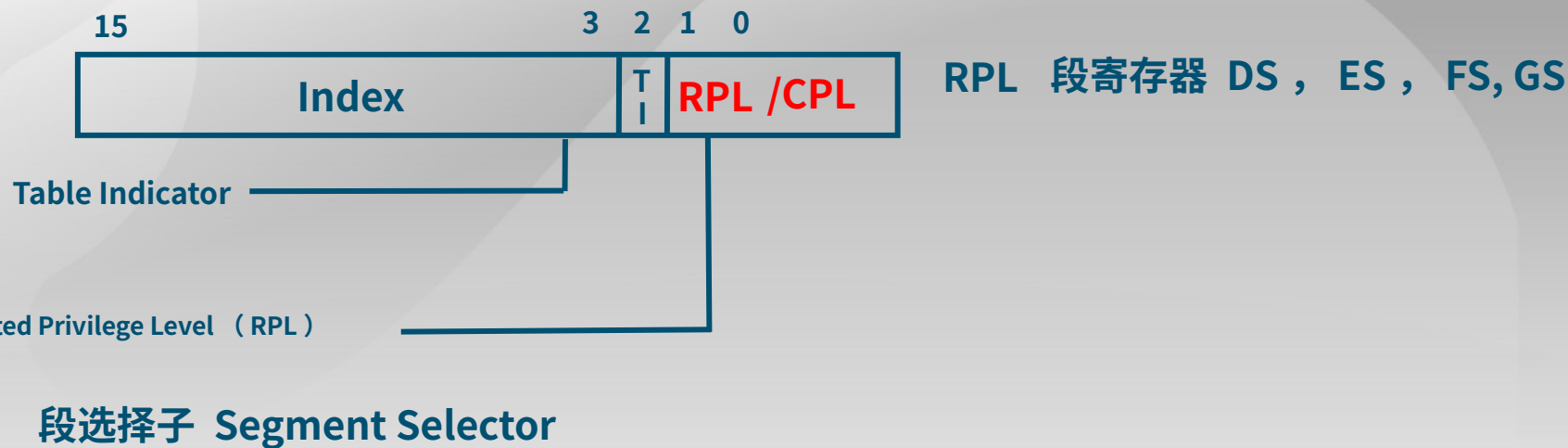
Trap Gate



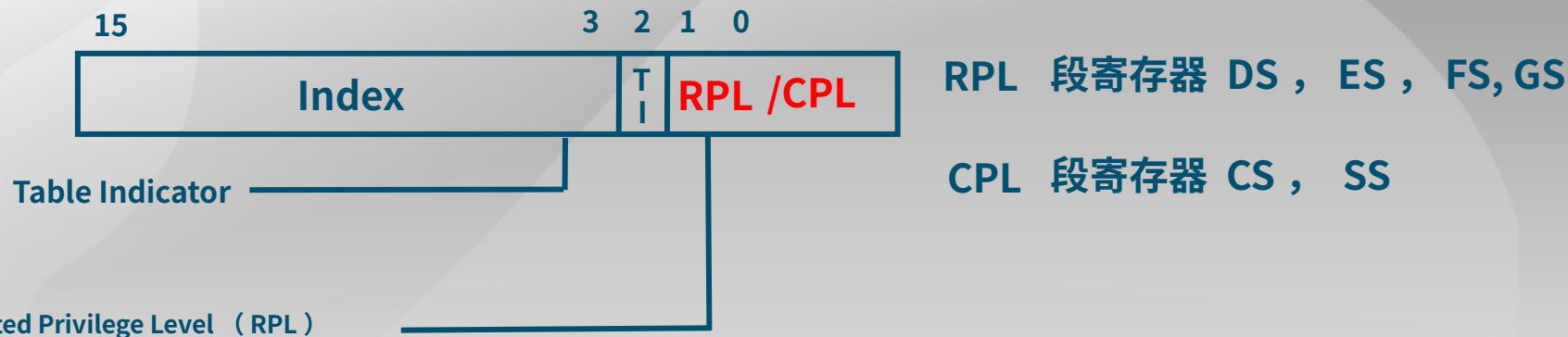
x86 特权级 - 特权转移



x86 特权级 - 当前的特权级 ?



x86 特权级 - 当前的特权级 ?



段选择子 Segment Selector

x86 特权级 – 当前的特权级？

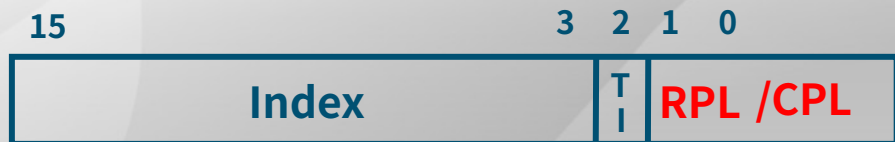


Table Indicator

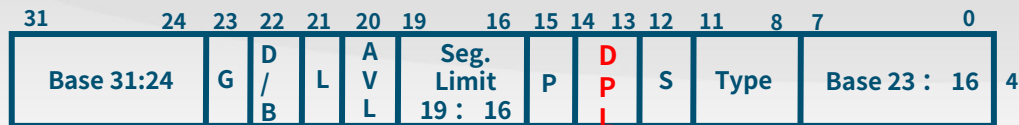
Requested Privilege Level (RPL)

RPL 段寄存器 DS , ES , FS, GS

CPL 段寄存器 CS , SS

DPL 段描述符, 门描述符

段选择子 Segment Selector



段 / 门描述符

x86 特权级 –当前的特权级？

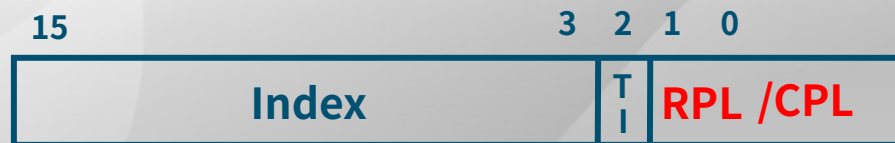


Table Indicator

Requested Privilege Level (RPL)

RPL 段寄存器 DS , ES , FS, GS

CPL 段寄存器 CS , SS

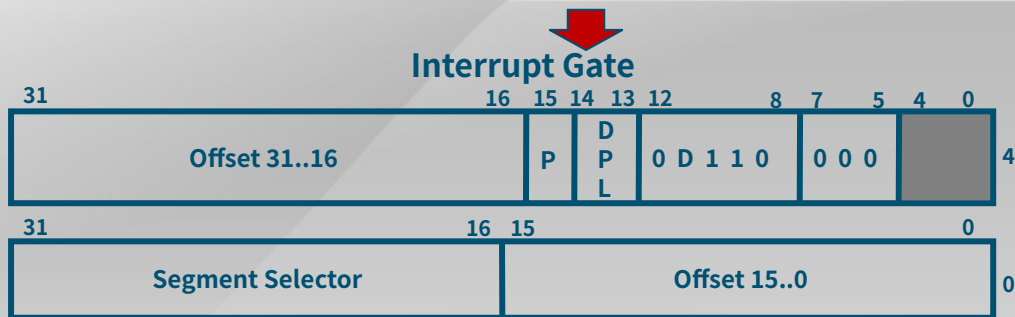
DPL 段描述符, 门描述符

段选择子 Segment Selector

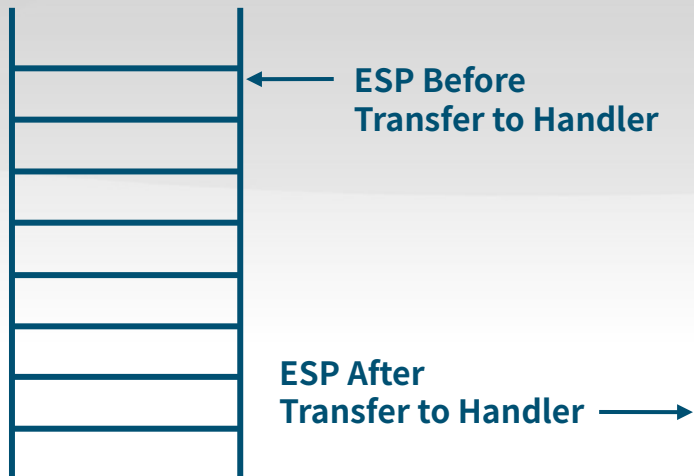
访问门时 $CPL \leq DPL[\text{门}] \ \& \ CPL \geq DPL[\text{段}]$

访问段时 $MAX(CPL, RPL) \leq DPL[\text{段}]$

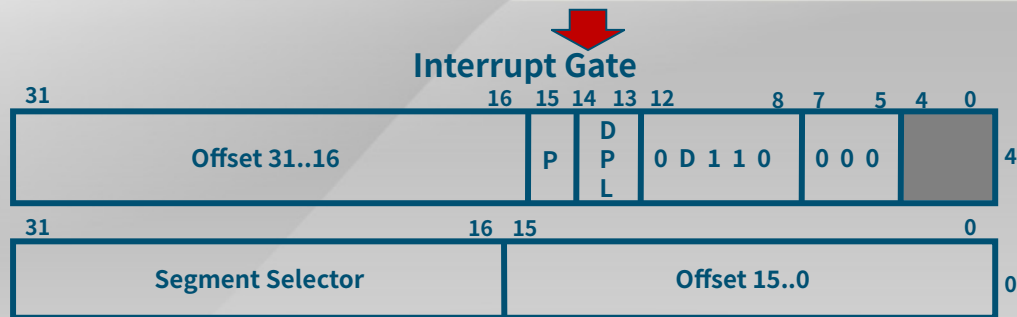
x86 特权级 - 通过中断切换特权级



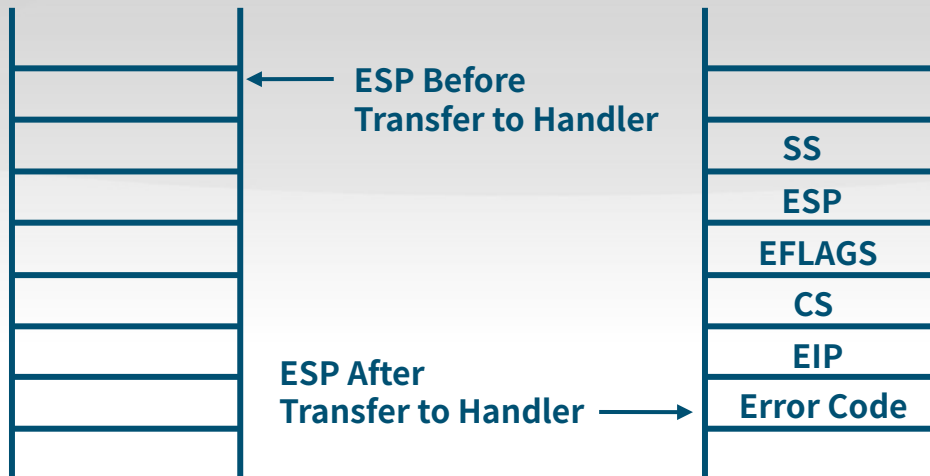
**Interrupt Stack Usage with
Privilege-Level Change**



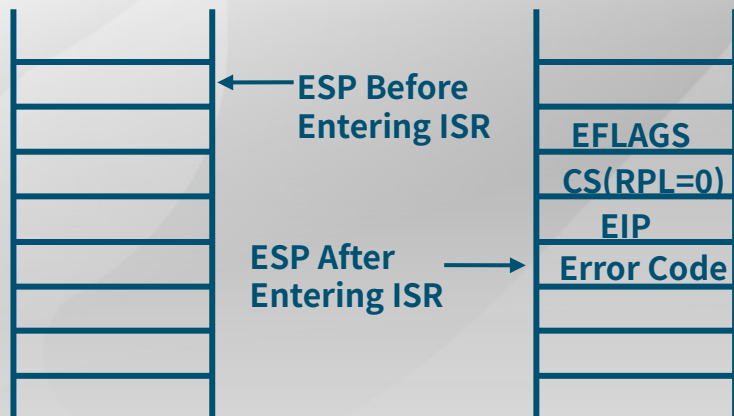
x86 特权级 - 通过中断切换特权级



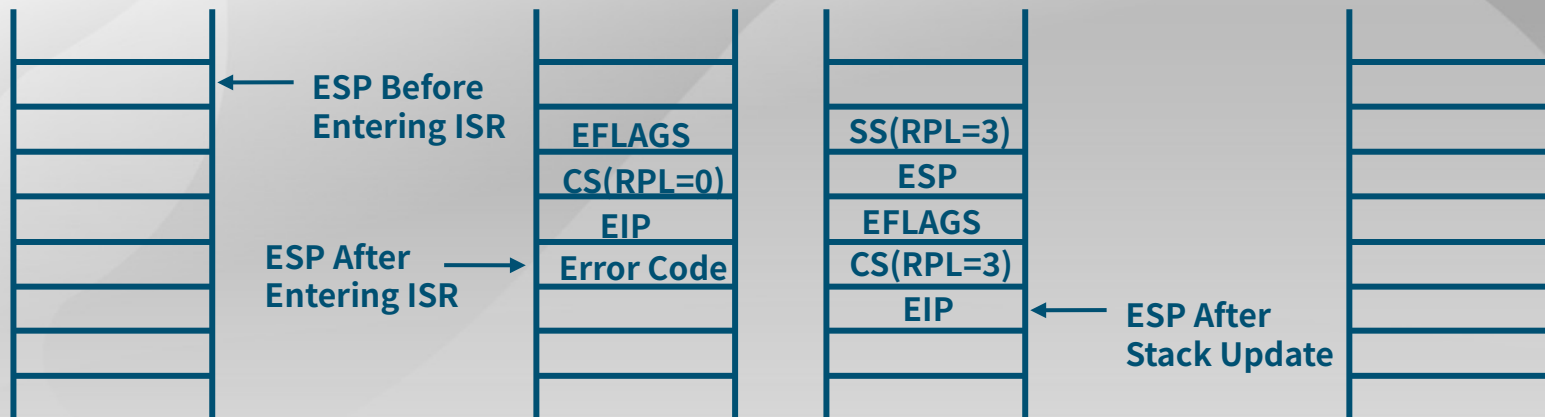
**Interrupt Stack Usage with
Privilege-Level Change**



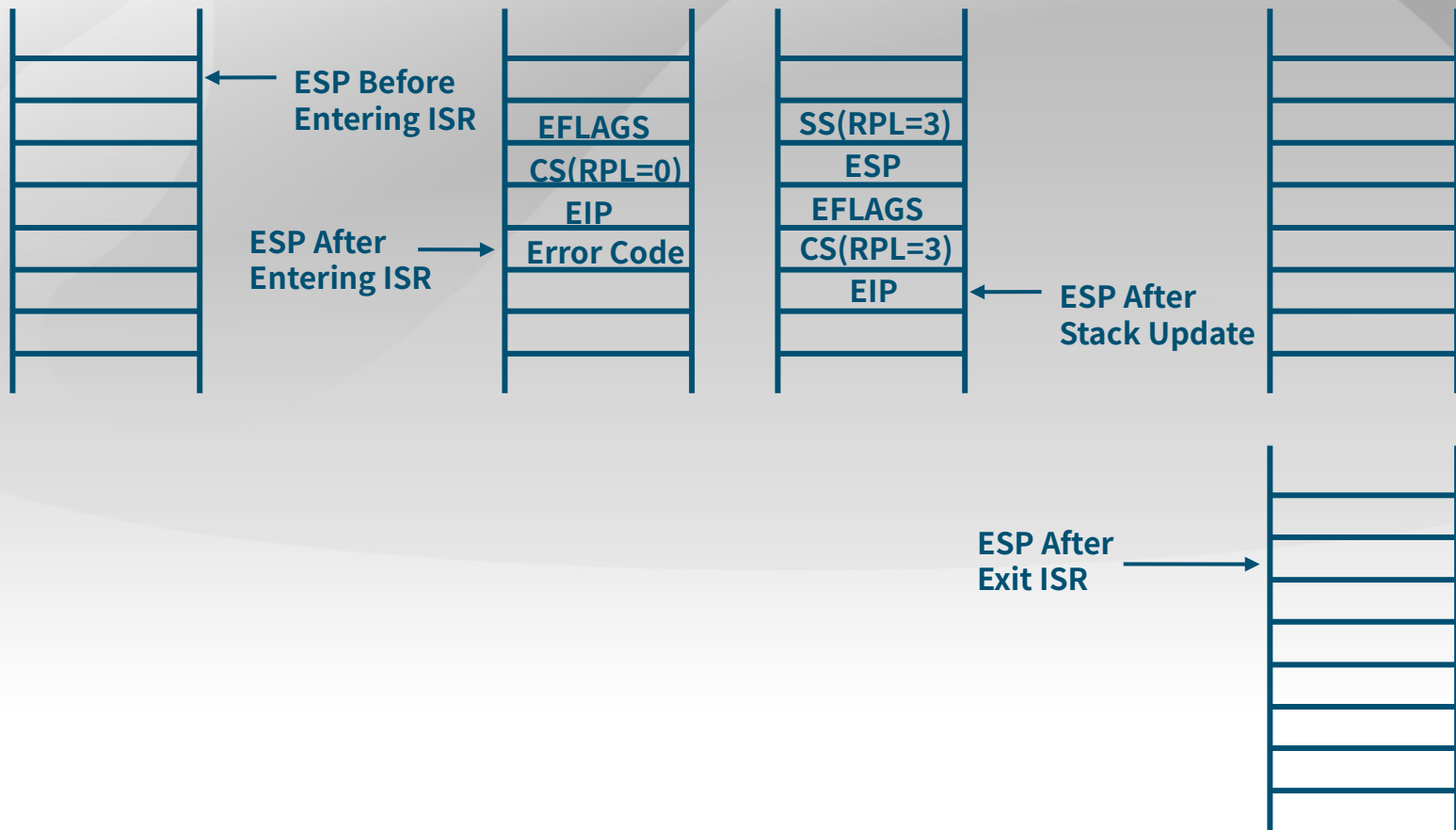
x86 特权级 - 切换特权级 (0 to 3)



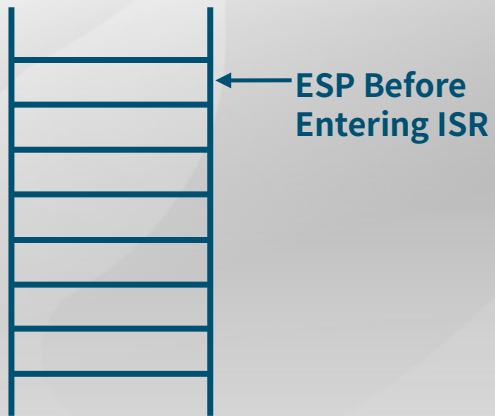
x86 特权级 - 切换特权级 (0 to 3)



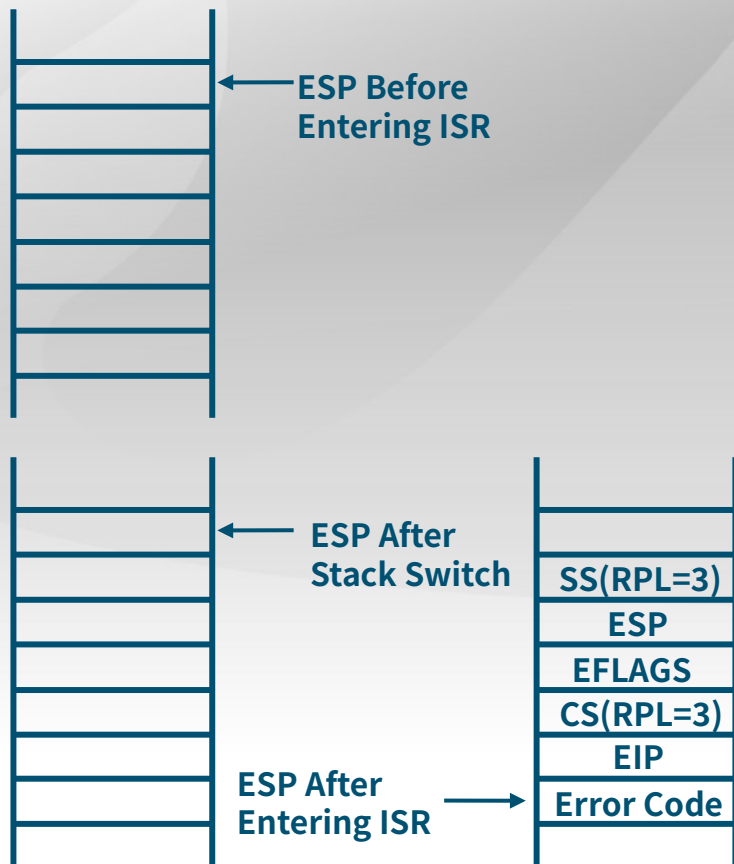
x86 特权级 - 切换特权级 (0 to 3)



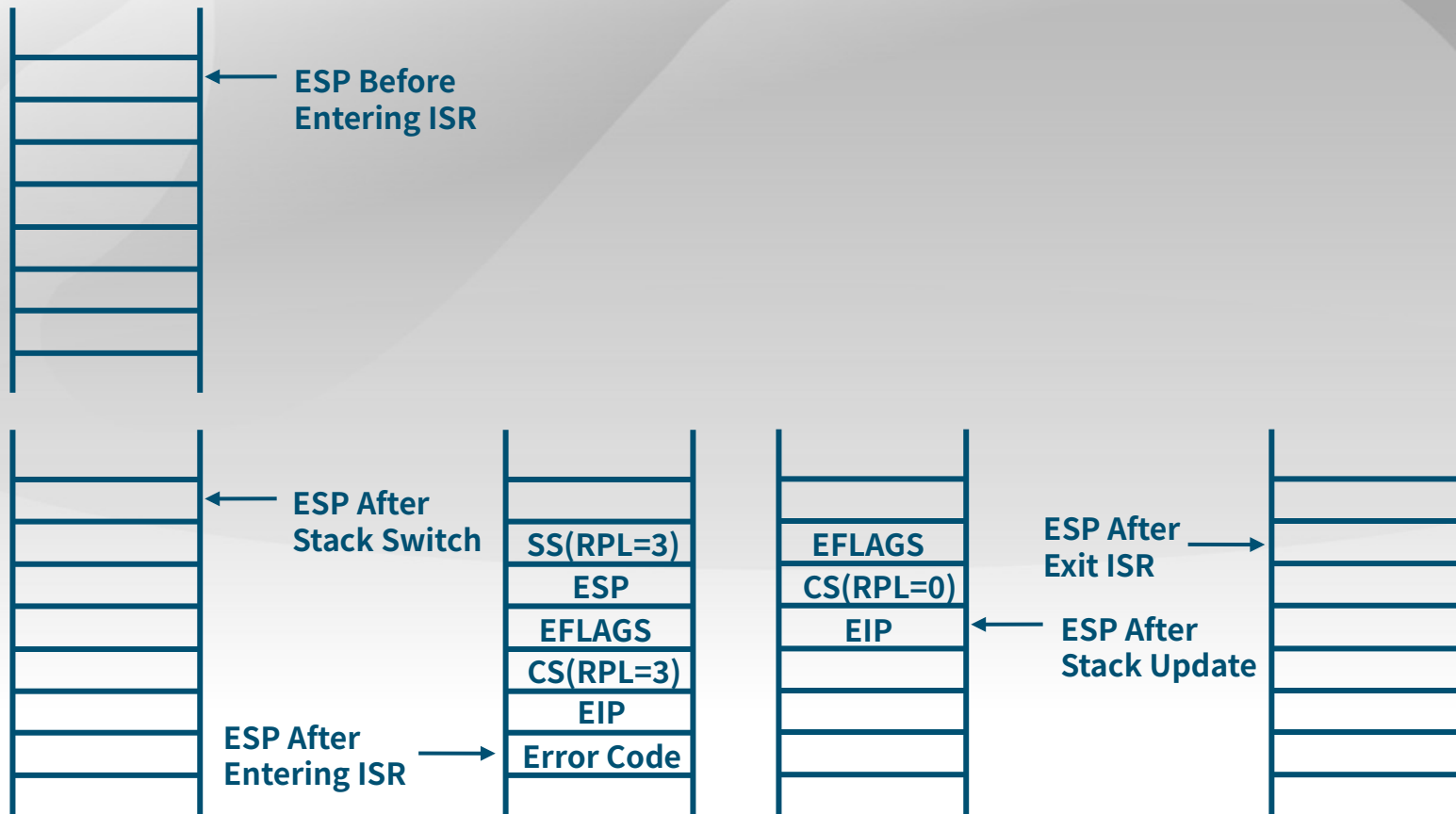
x86 特权级 - 切换特权级 (3 to 0)



x86 特权级 - 切换特权级 (3 to 0)



x86 特权级 - 切换特权级 (3 to 0)



x86 特权级 – TSS 格式

31	15	0	
I/O Map Base Address		Reserved	T 100
Reserved		LDT Segment Selector	96
Reserved		GS	92
Reserved		FS	88
Reserved		DS	84
Reserved		SS	80
Reserved		CS	76
Reserved		ES	72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3(PDBR)			28
Reserved		SS2	24
ESP2			20
Reserved		SS1	16
ESP1			12
Reserved		SS0	8
ESP0			4
Reserved		Previous Task Link	0

Reserved bits. Set to 0.

图 32-Bit Task-State Segment(TSS)

x86 特权级 – 栈切换中获取新的栈

■ TSS = Task State Segment



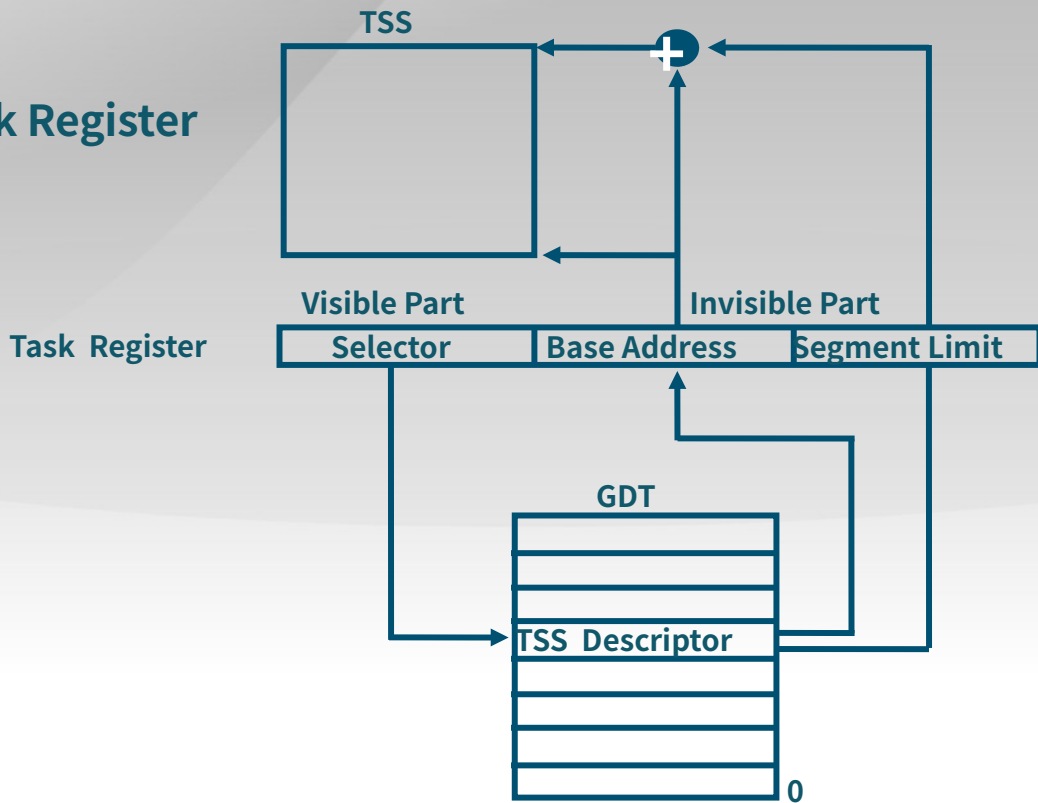
AVL Available for use by system software
B Busy flag
BASE Segment Base Address
DPL Descriptor privilege level
G Granularity
LIMIT Segment Limit
P Segment Present
TYPE Segment Type

图 Task State Segment 描述符 TSS Descriptor

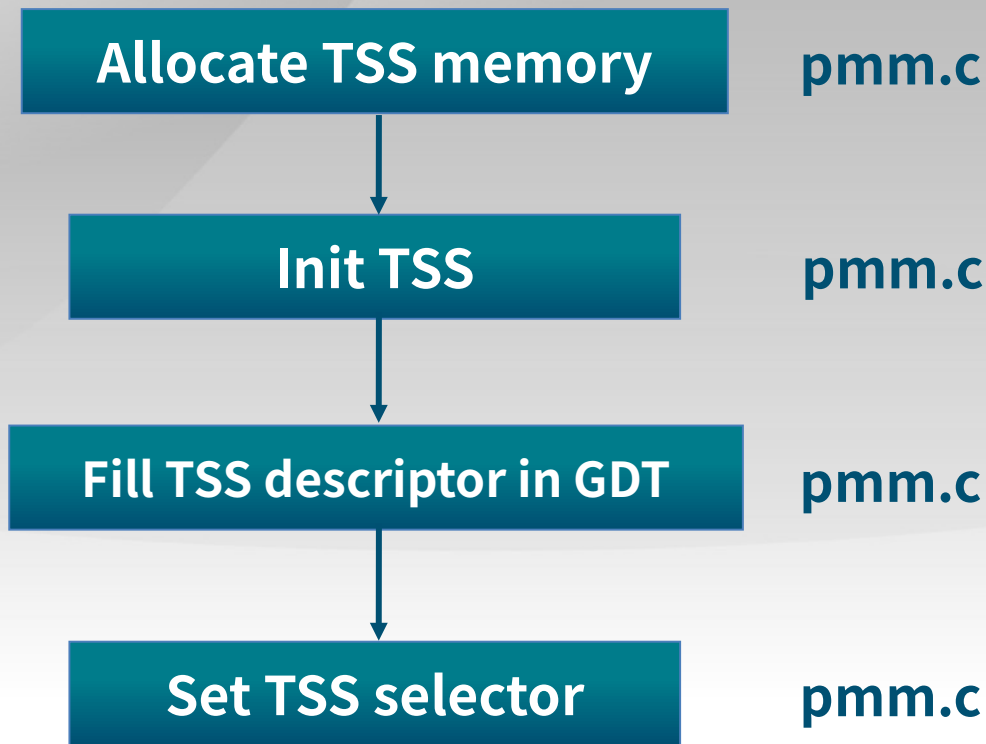
x86 特权级 - 栈切换中获取新的栈

■ TSS = Task State Segment

图 Task Register



x86 特权级 - 建立 TSS



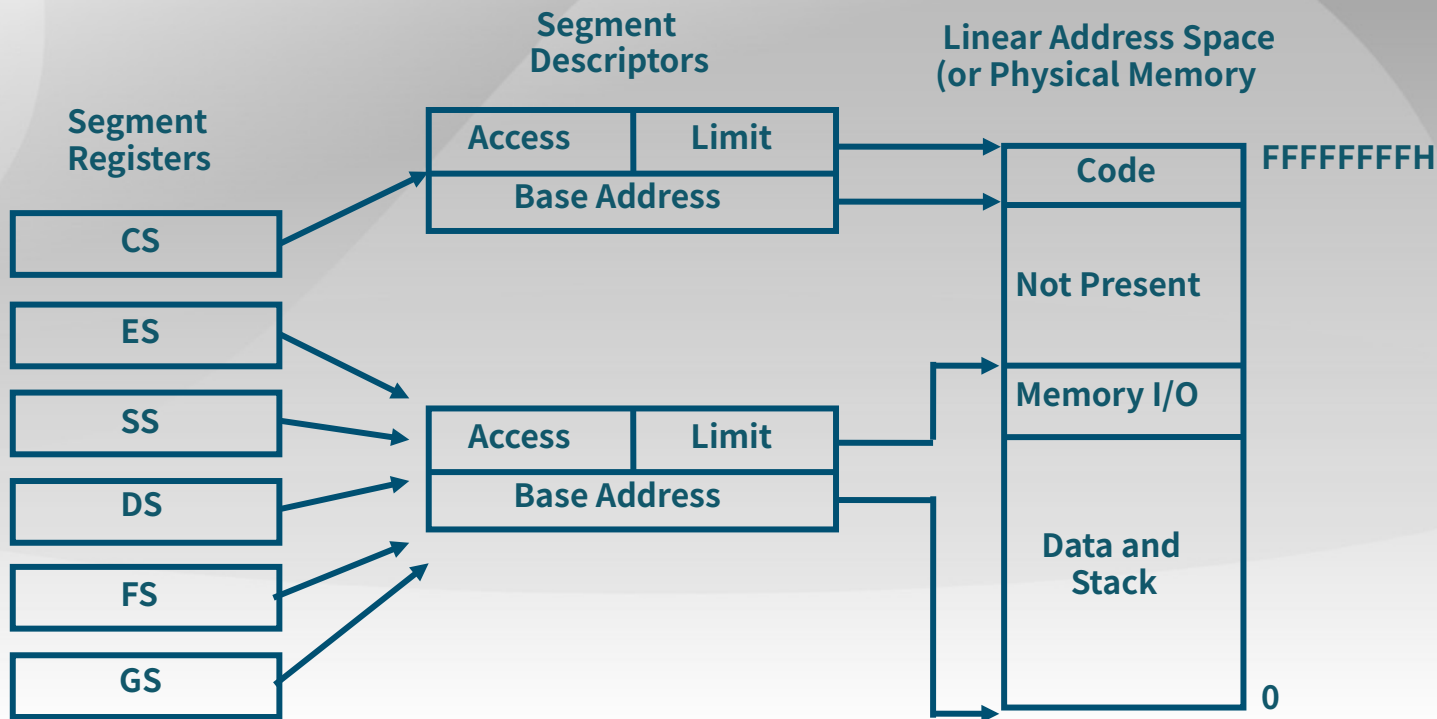
x86 特权级 – 参考文献

- Chap. 6.3.5, Vol. 1, Intel® and IA-32 Architectures Software Developer's Manual
- Chap. 7, Vol. 3, Intel® and IA-32 Architectures Software Developer's Manual

- 了解页表格式
- 了解如何建立段表 + 页表
- 了解如何操作页表项

X86 内存管理单元 MMU

x86 MMU – 段机制概述



Protected Flat Model

x86 MMU – 段选择子（segment selector）中的隐藏部分

Visible Part	Hidden Part	
Segment Selector	Base Address,Limit,Access Information	
		CS
		SS
		DS
		ES
		FS
		GS

图 Segment Registers

- 基址（Base address）一直被存放在隐藏部分。直到选择子发生变化，才会更新基址内容（即新的段表项中的基址值）。

x86 hardware MMU – 建立 GDT tables (kernel init)

Init GDT table as an array

entry.S (base address is -0xC0000000)

Init GDT descriptor

entry.S

Invoking lgdt

entry.S

Update DS, ES, SS, etc.

entry.S

Update CS using ljmp

entry.S

x86 MMU – 建立 GDT tables (bootloader)

Init GDT table as an array

bootasm.S (base address is 0x0)

Init GDT descriptor

bootasm.S

Invoking lgdt

bootasm.S

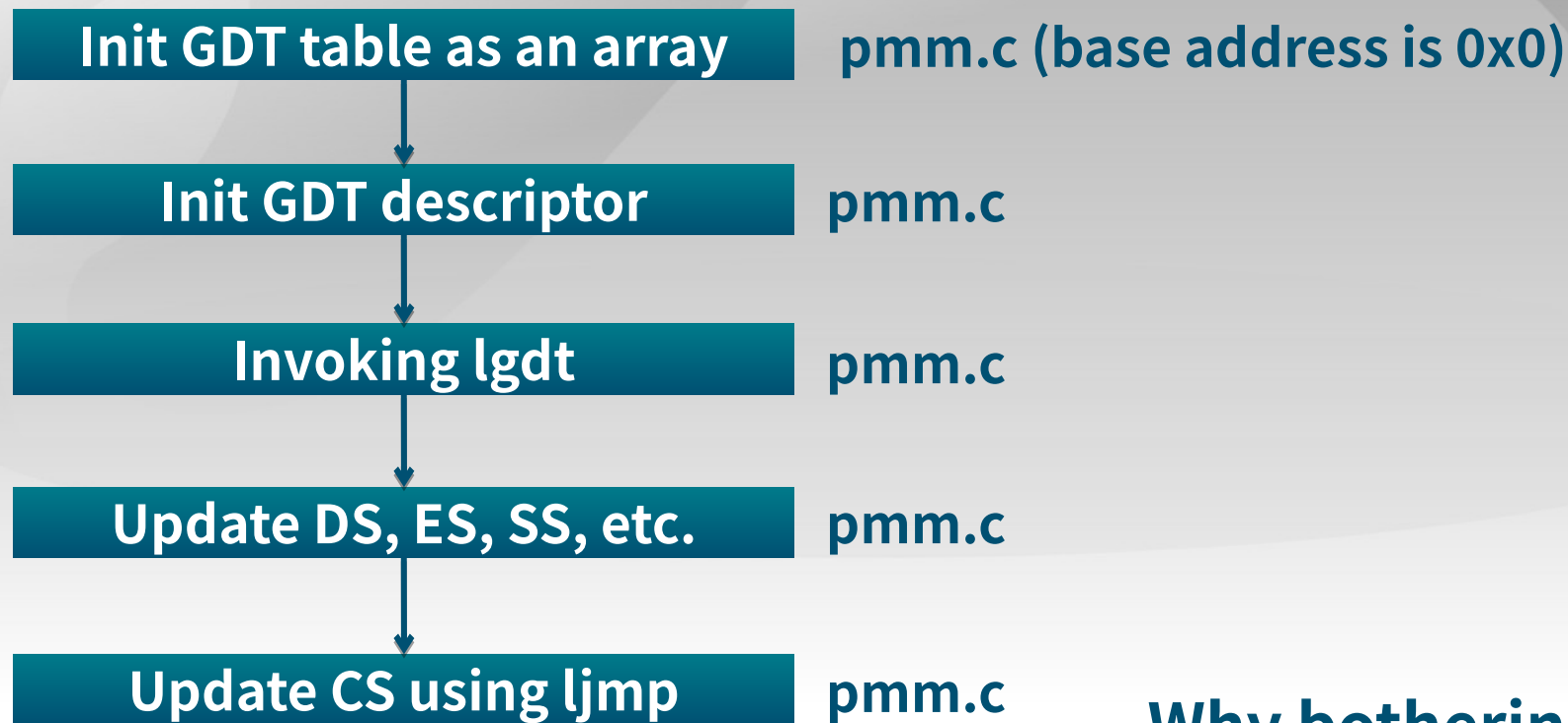
Set bit 0 in CR0

bootasm.S

Update CS using ljmp

bootasm.S

x86 MMU – 建立 GDT tables (使能页机制 enable paging)



Why bothering?!

x86 MMU – 页机制概述

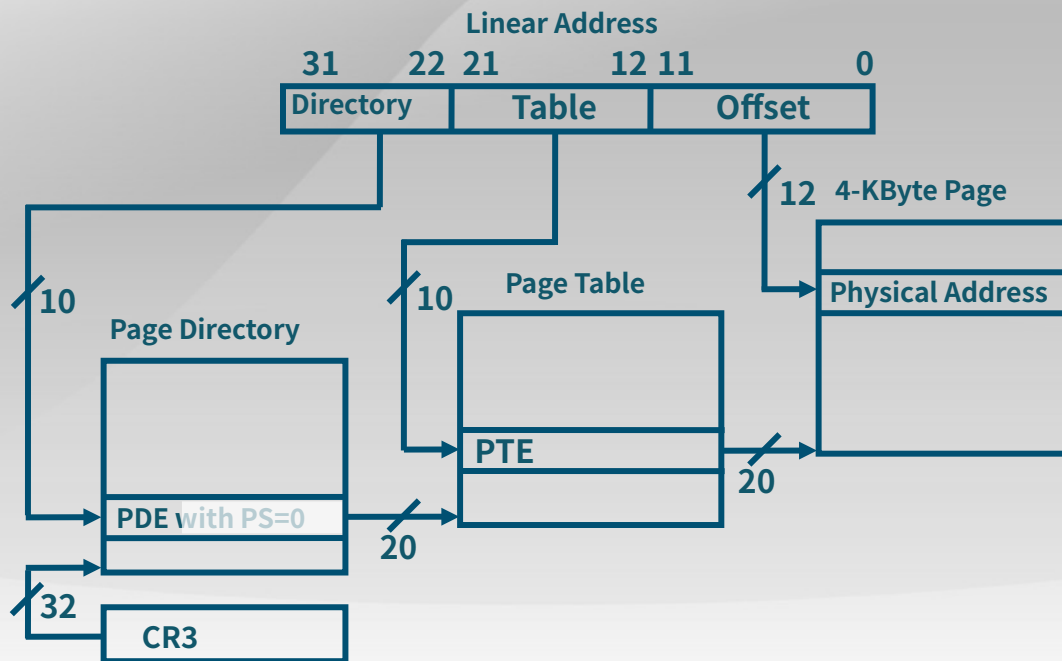
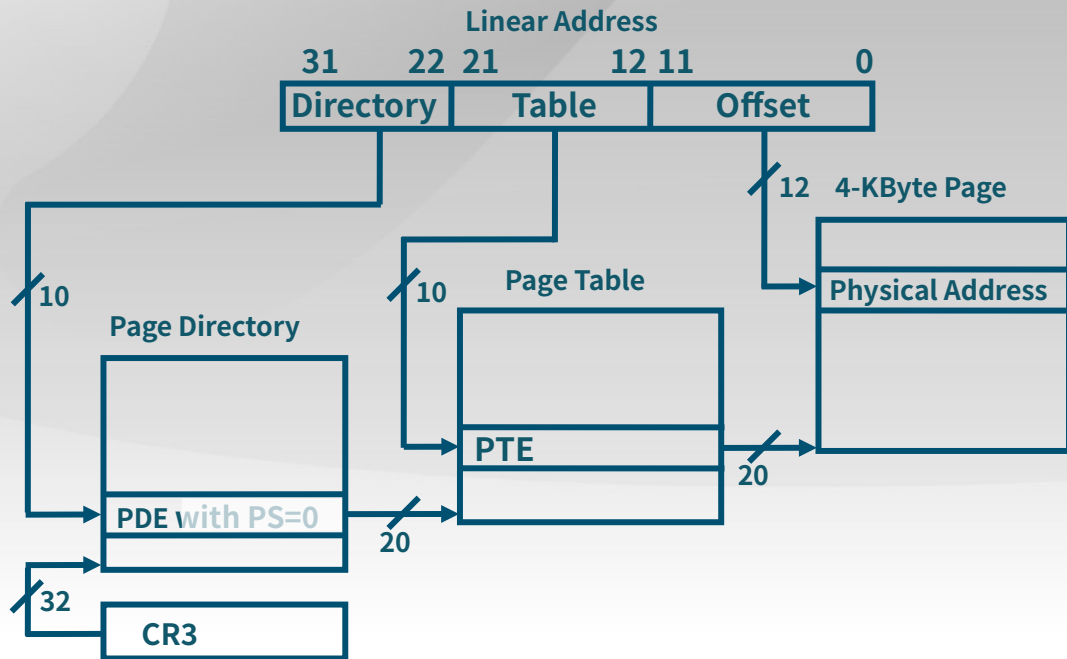


图 基于页机制的地址转换

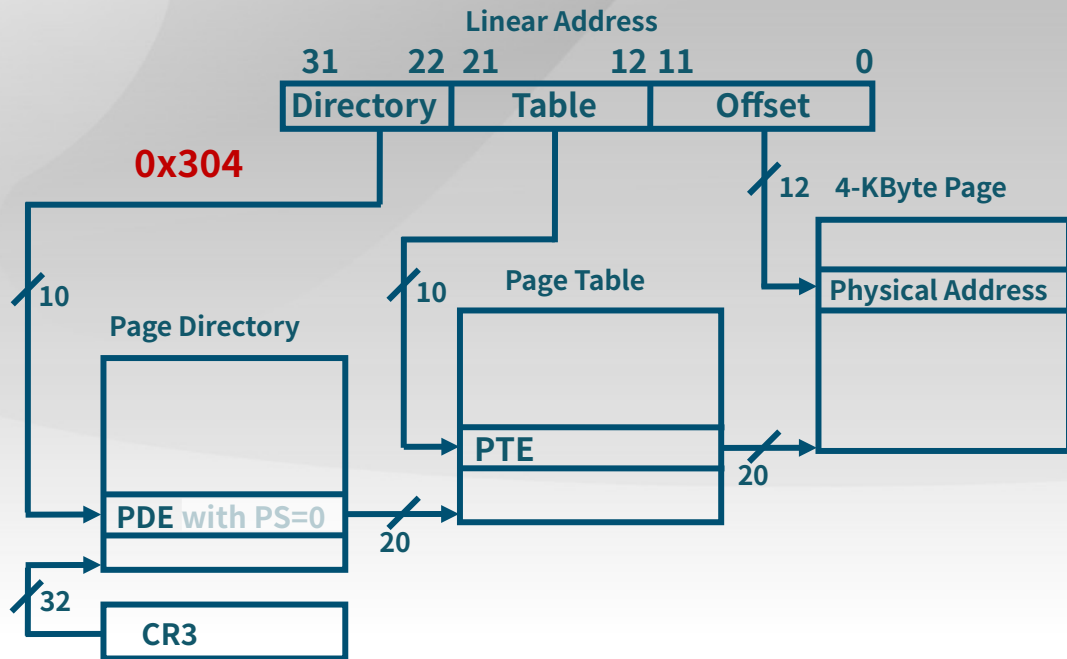
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



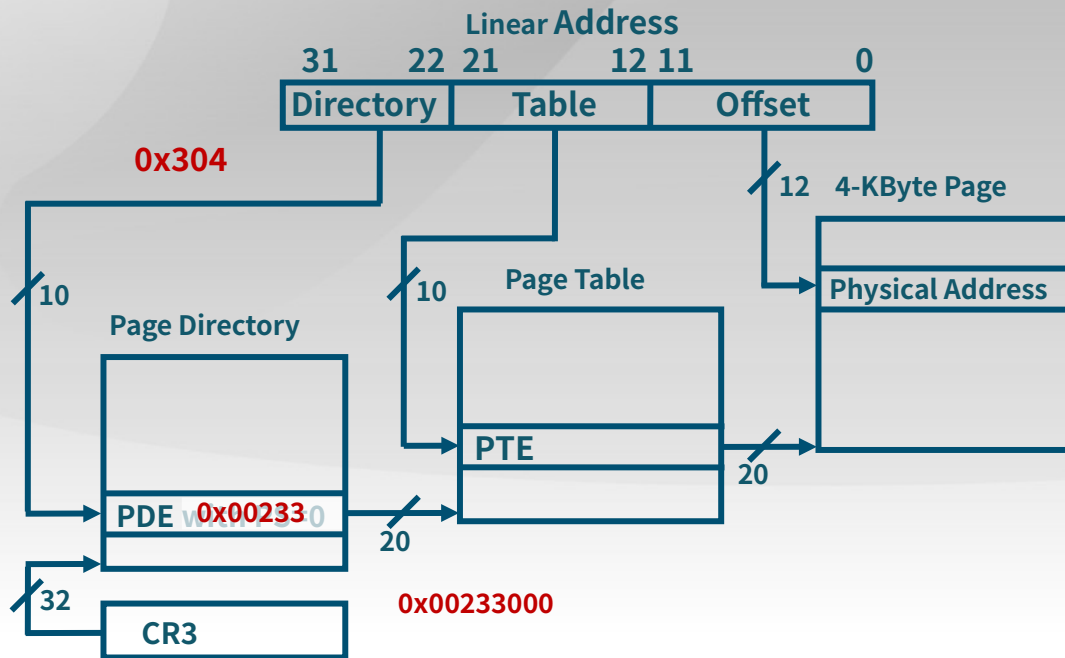
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



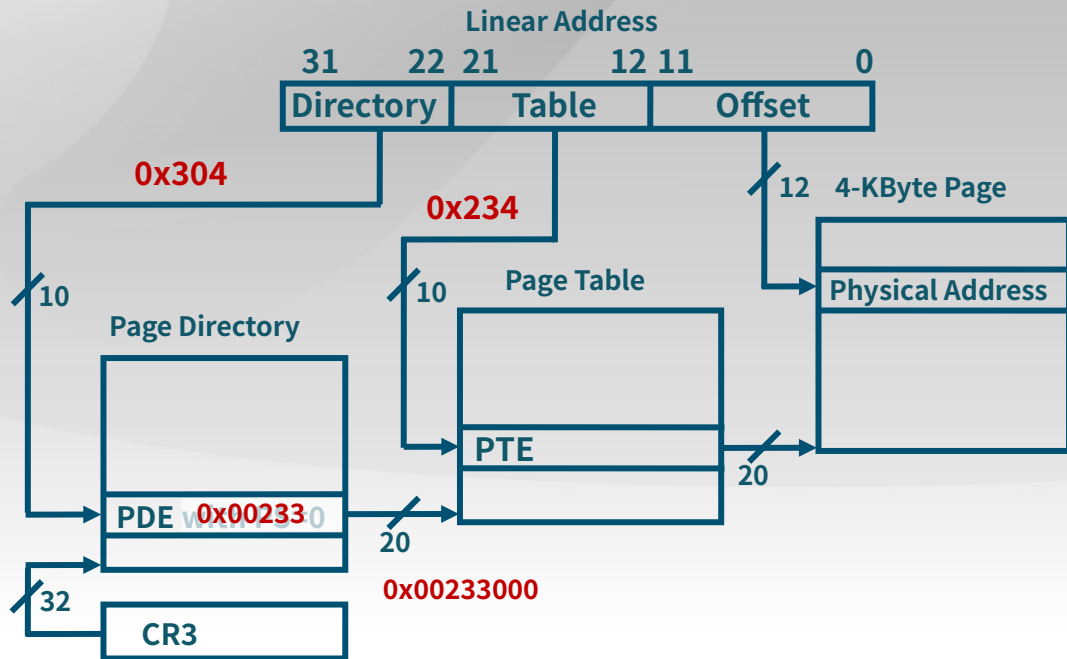
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



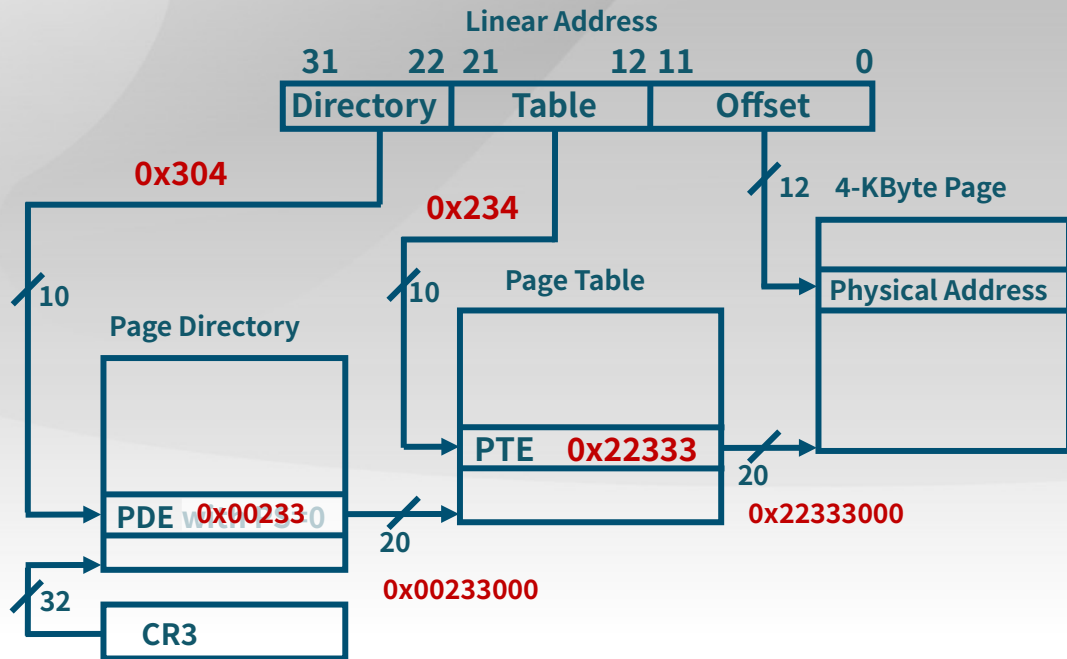
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



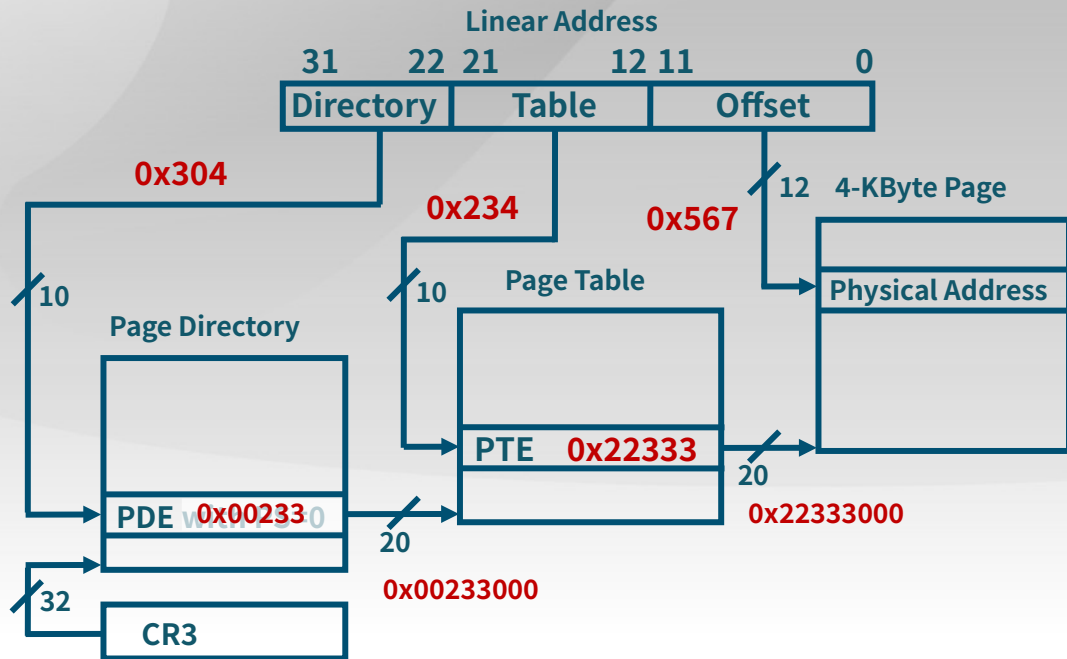
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



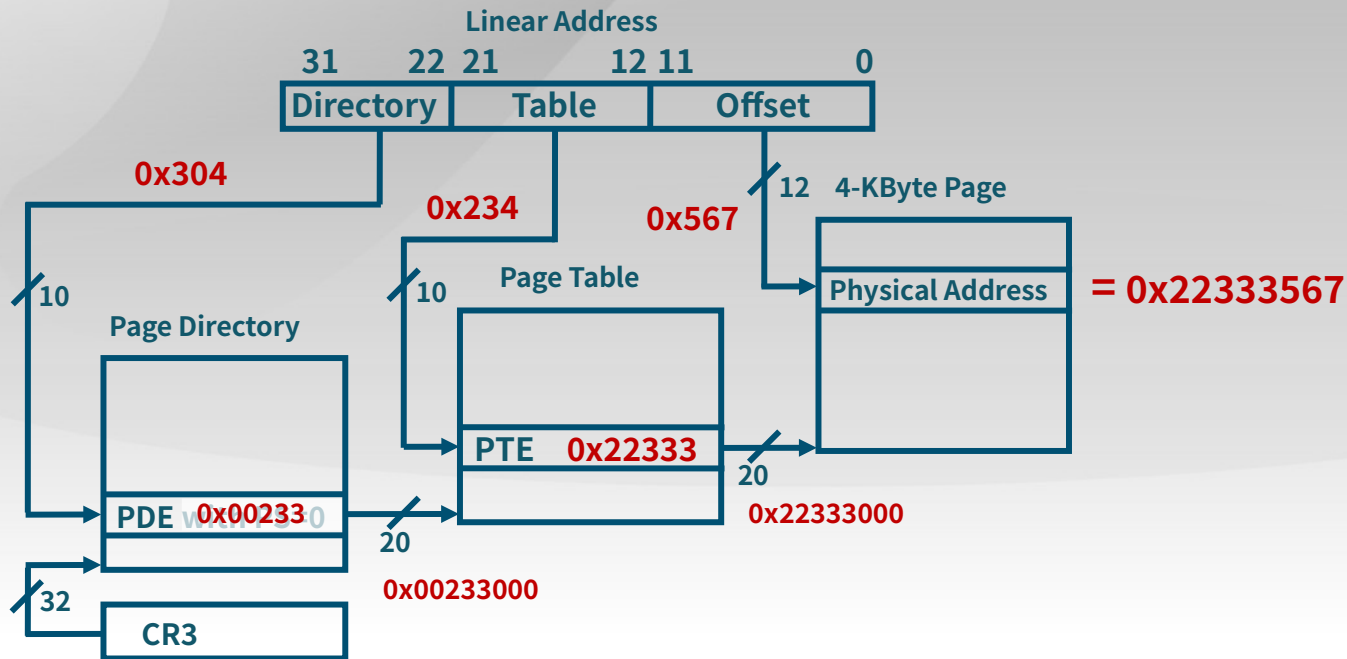
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



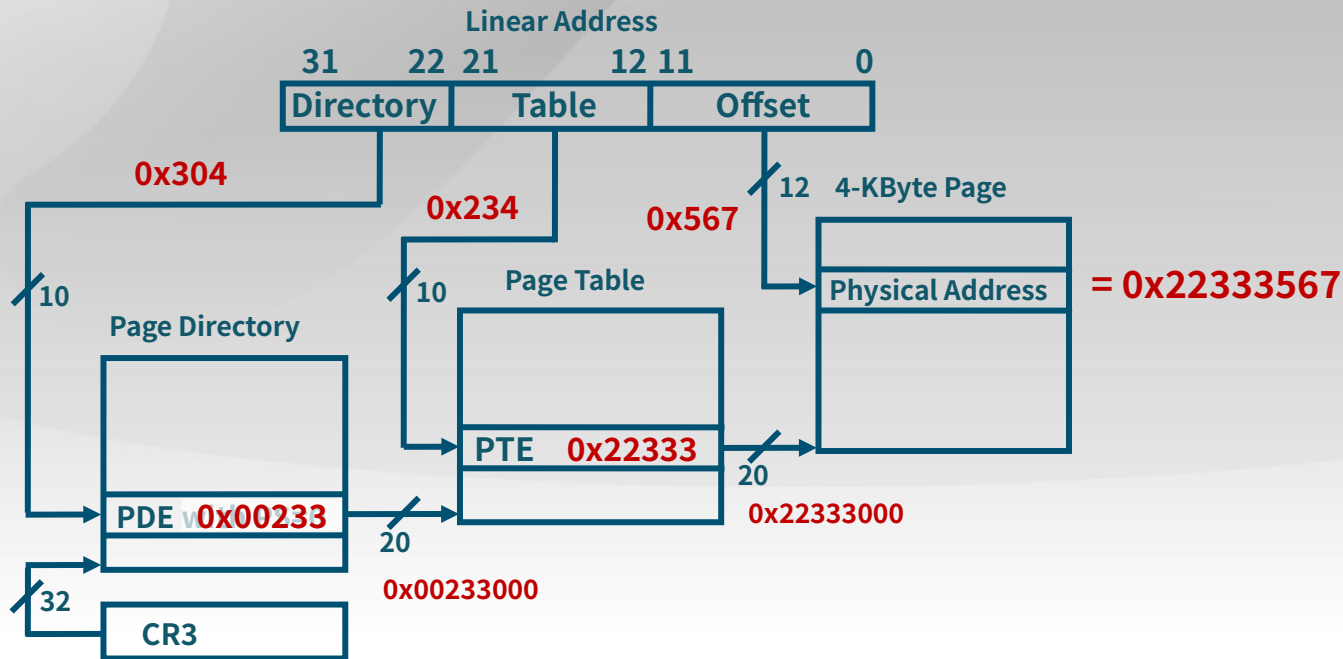
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567



在页表项中存放的地址内容是线性地址（linear addresses）！

x86 MMU – 页表项 (page table entries)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Address of page directory ¹																				Ignored					P C D	P W T	Ignored			CR3			
Bits 31:22 of address of 2MB page frame										Reserved (must be 0)					Bits 39:32 of address ²		P A T	Ignored	G	1	D	A	P C D	P W T					PDE: 4MB page				
Address of page table																				Ignored					0	I g n	A	P C D	P W T				PDE: page table
Ignored																											PDE: not present						
Address of 4KB page frame																				Ignored	G	P A T	D	A	P C D	P W T				PTE: 4KB page			
Ignored																											PTE: not present						

- R/W: 1 if this page is writable
- U/S: 1 if this page is accessible in ring 3
- A: 1 if this page has been accessed
- You may ignore others for now

图 页目录项和页表项的结构

x86 MMU – 使能页机制 (enable paging)

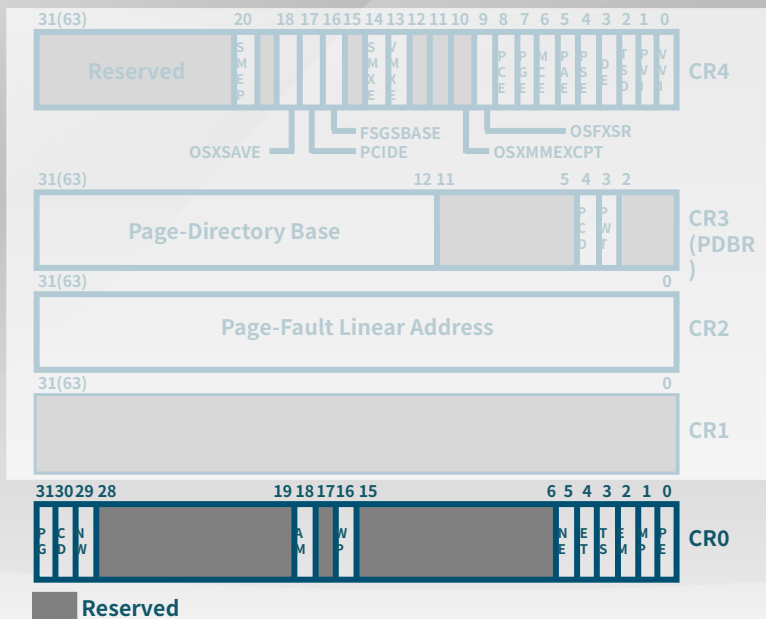


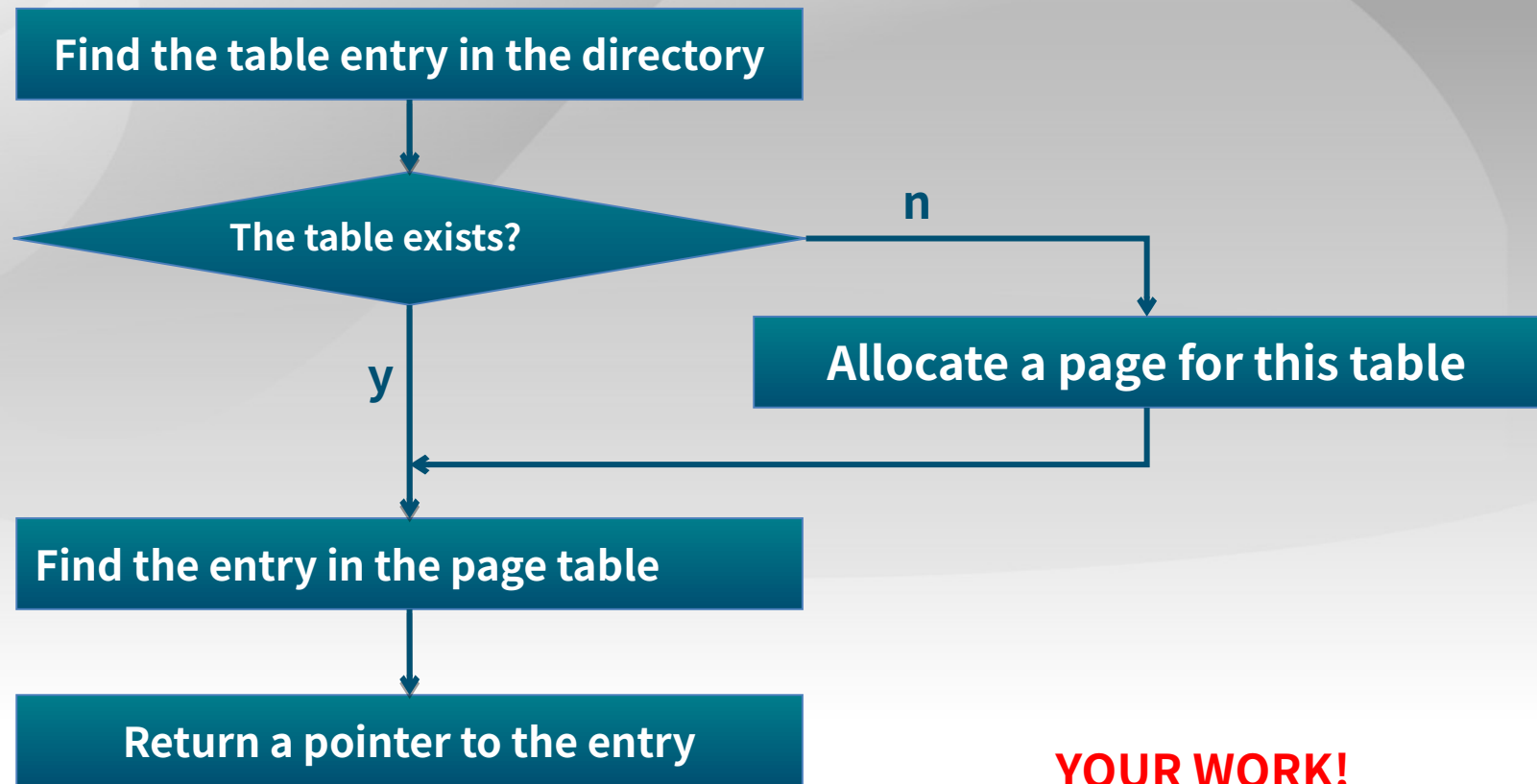
图 Control Registers

为了在保护模式下使能页机制，OS 需要置 CR0 寄存器中的 bit 31 (PG)

x86 MMU – 建立页表 (page tables)



x86 MMU – 在页表中建立页的映射关系



YOUR WORK!

x86 MMU – 合并段机制 + 页机制 (segmentation + paging)

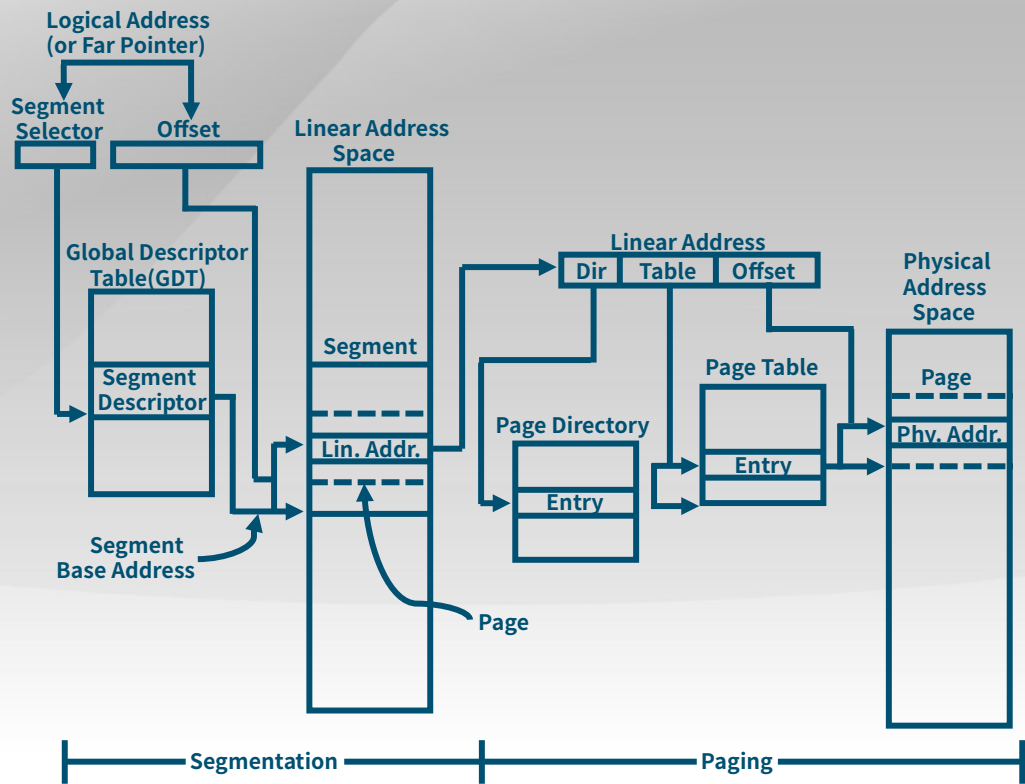
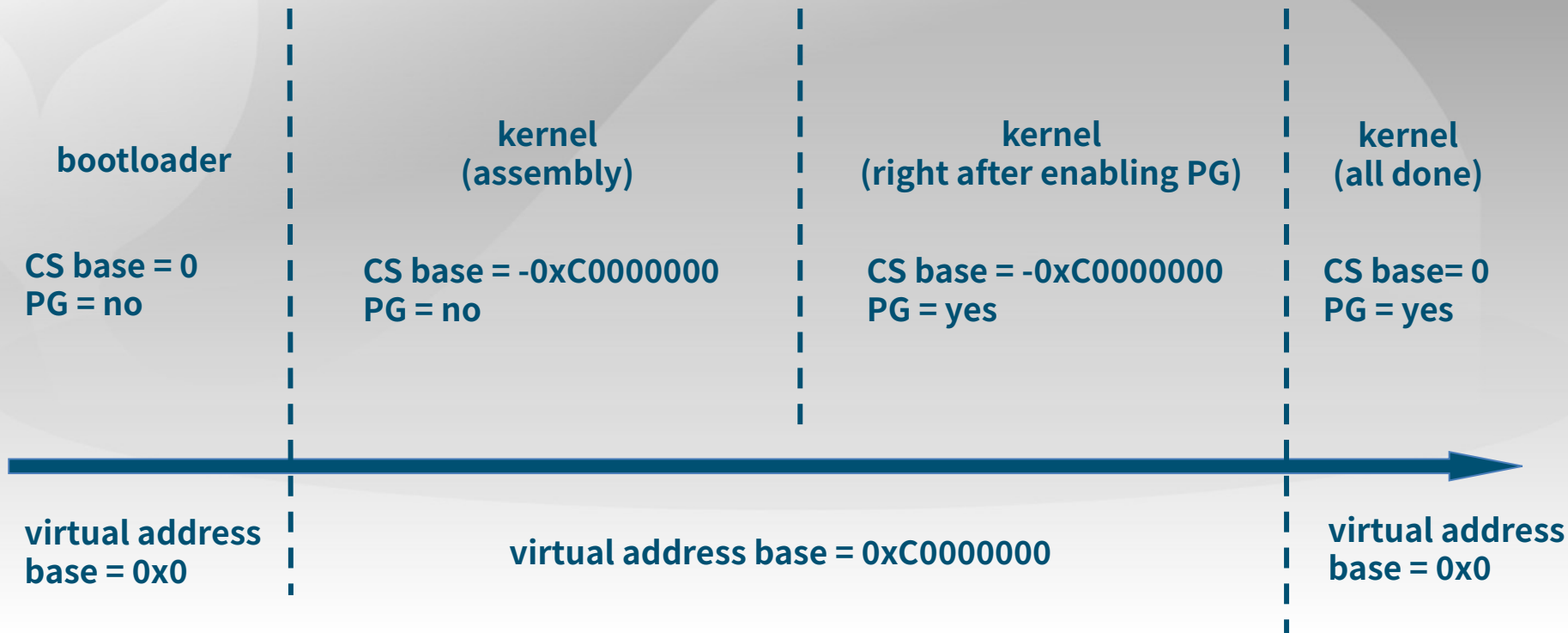


图 段页式内存映射机制 Segmentation and Paging

x86 MMU – uCore 内存管理初始化



x86 MMU – 参考资料

- Chap. 3 & 4, Vol. 3, Intel® and IA-32 Architectures Software Developer's Manual