操作系统课程实验
Lab2：物理内存管理

陈 渝

清华大学计算机系

2014年秋季

大纲

- 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

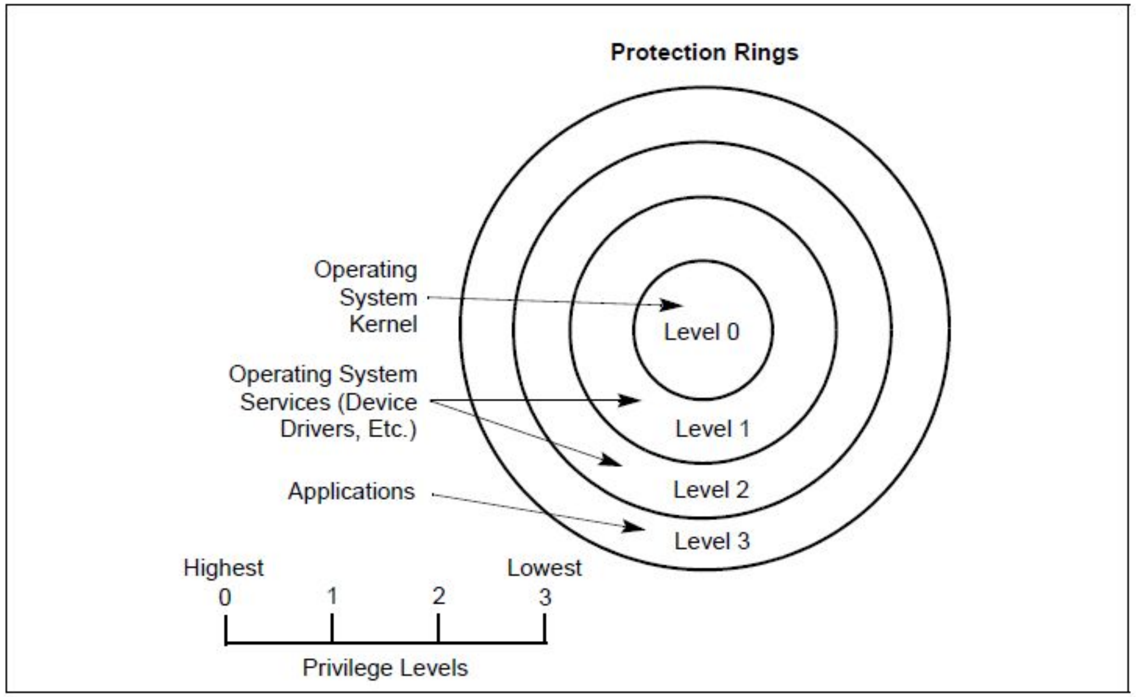


Figure 6-3. Protection Rings

- 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 – 当前的特权级?

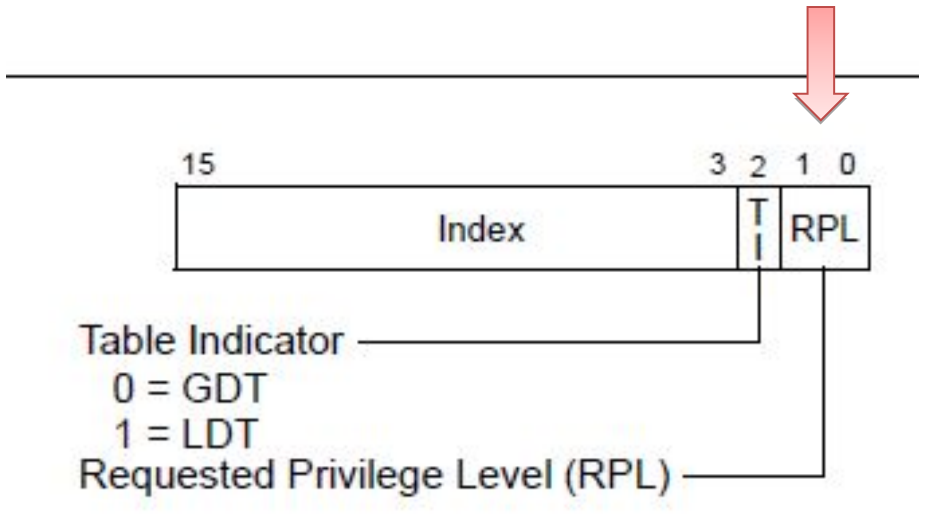


Figure 3-6. Segment Selector

x86 特权级 –特权级检查举例

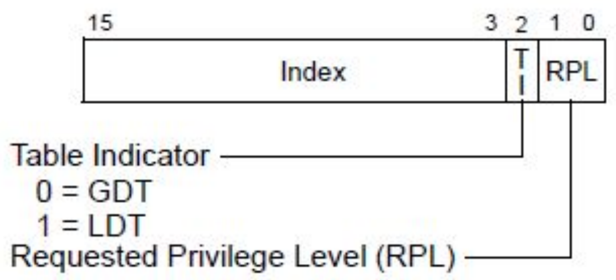
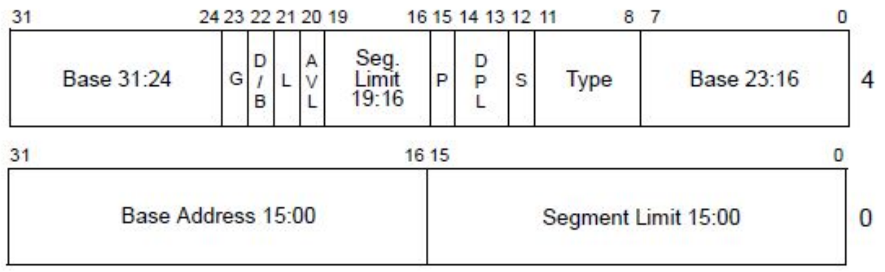


Figure 3-6. Segment Selector



L	— 64-bit code segment (IA-32e mode only)
AVL	— Available for use by system software
BASE	— Segment base address
D/B	— Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL	— Descriptor privilege level
G	— Granularity
LIMIT	— Segment Limit
P	— Segment present
S	— Descriptor type (0 = system; 1 = code or data)
TYPE	— Segment type

Figure 3-8. Segment Descriptor

x86 特权级 –特权级检查举例

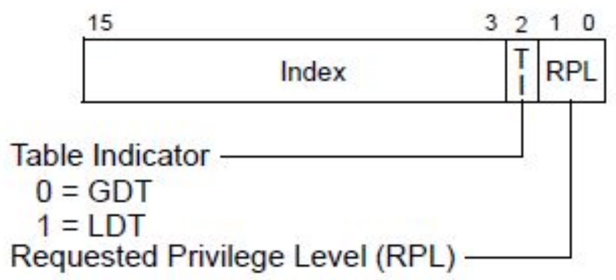
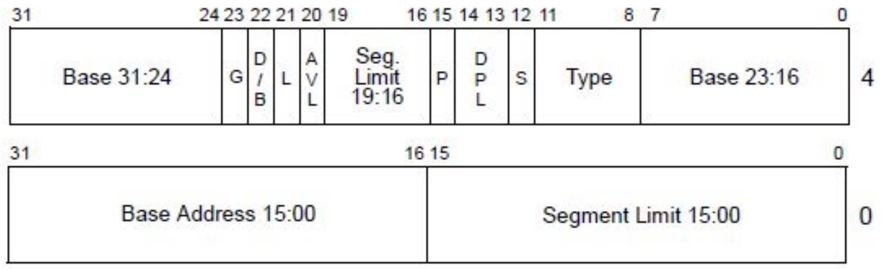
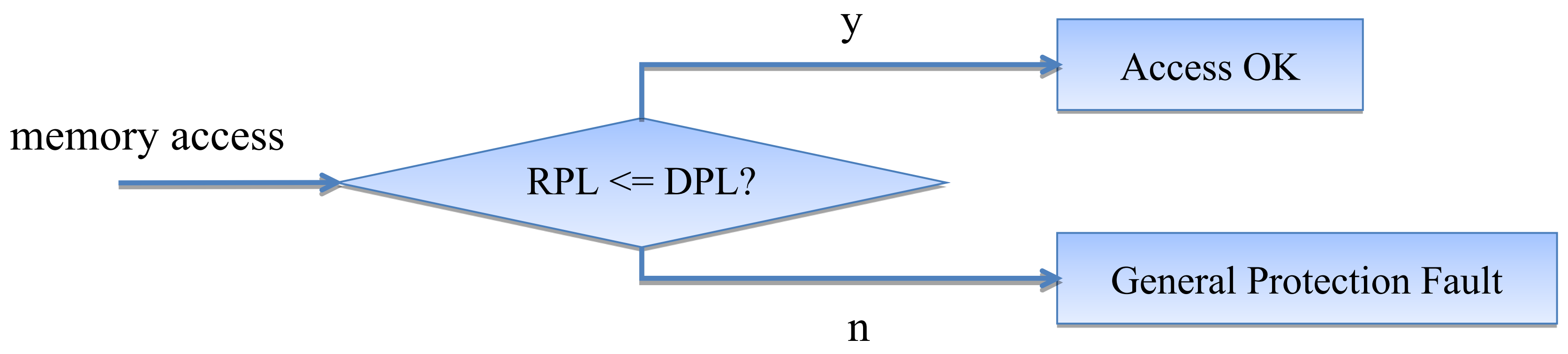


Figure 3-6. Segment Selector

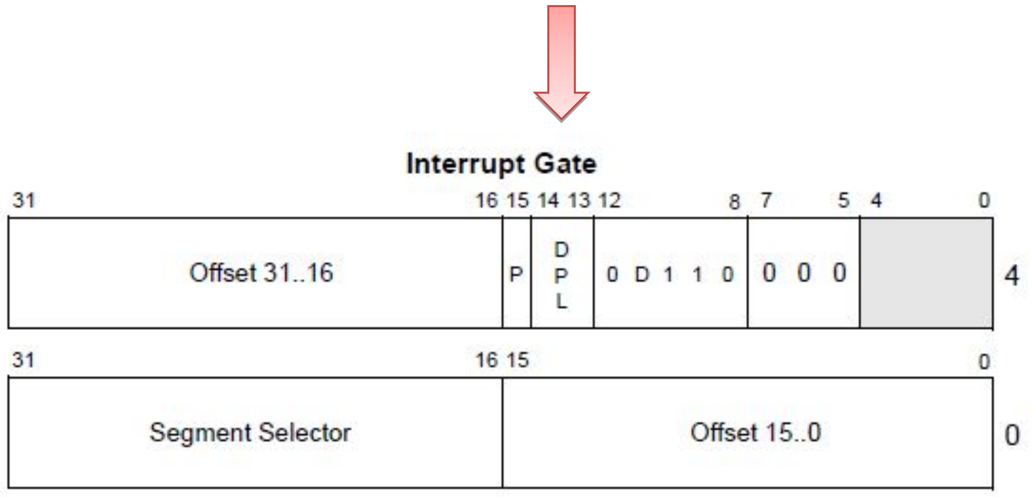


L	— 64-bit code segment (IA-32e mode only)
AVL	— Available for use by system software
BASE	— Segment base address
D/B	— Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL	— Descriptor privilege level
G	— Granularity
LIMIT	— Segment Limit
P	— Segment present
S	— Descriptor type (0 = system; 1 = code or data)
TYPE	— Segment type

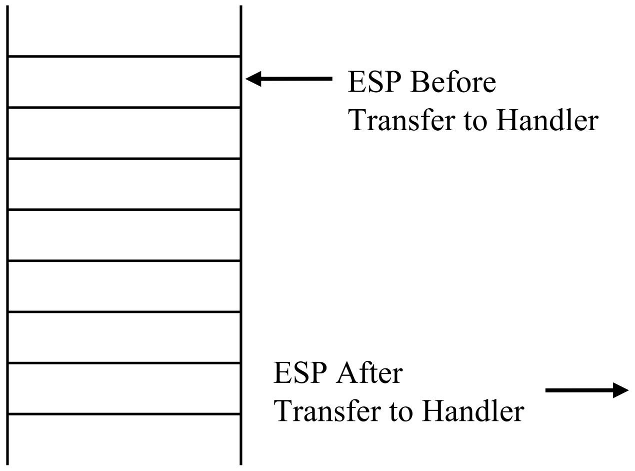
Figure 3-8. Segment Descriptor



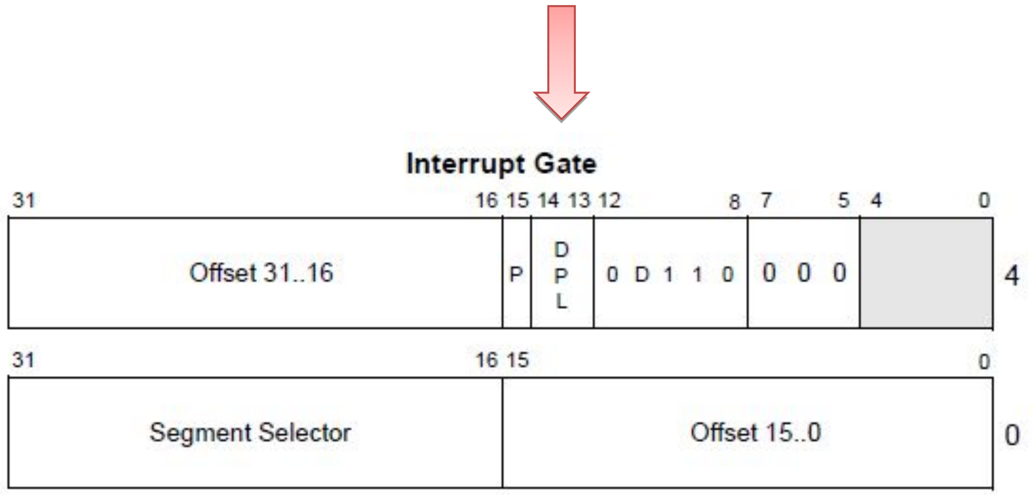
x86 特权级 –通过中断切换特权级



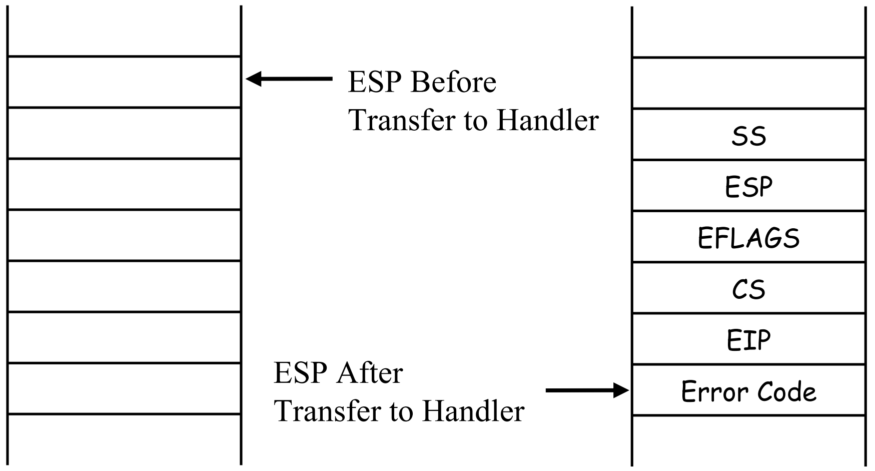
Interrupt Stack Usage with Privilege-Level Change



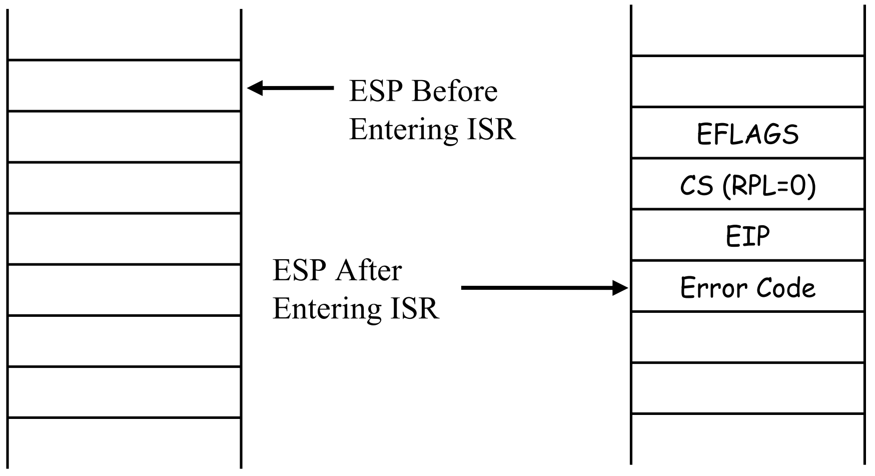
x86 特权级 –通过中断切换特权级



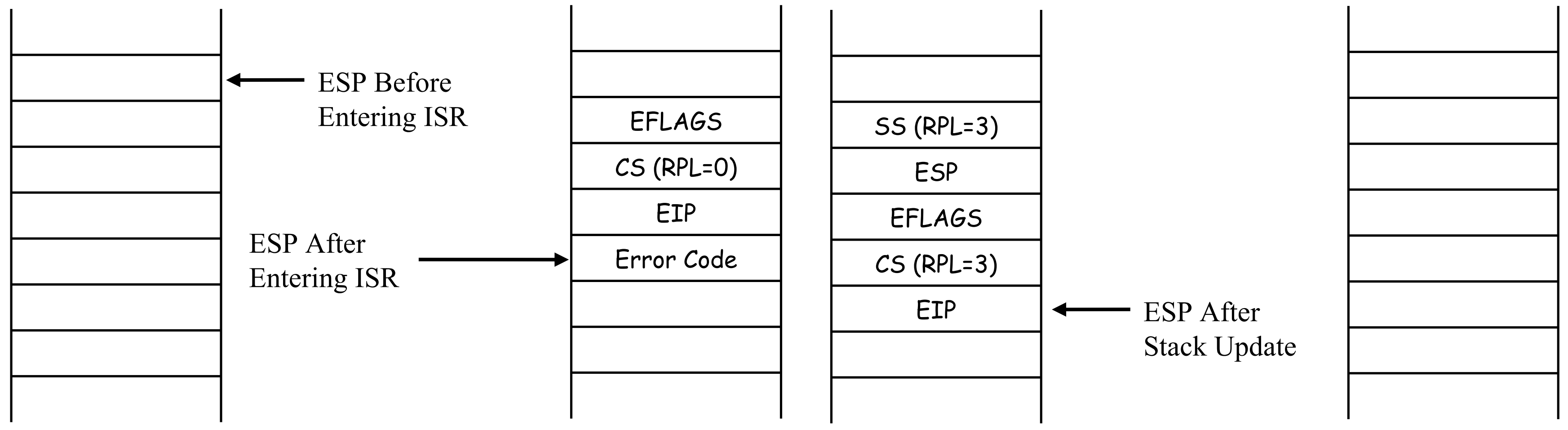
Interrupt Stack Usage with Privilege-Level Change



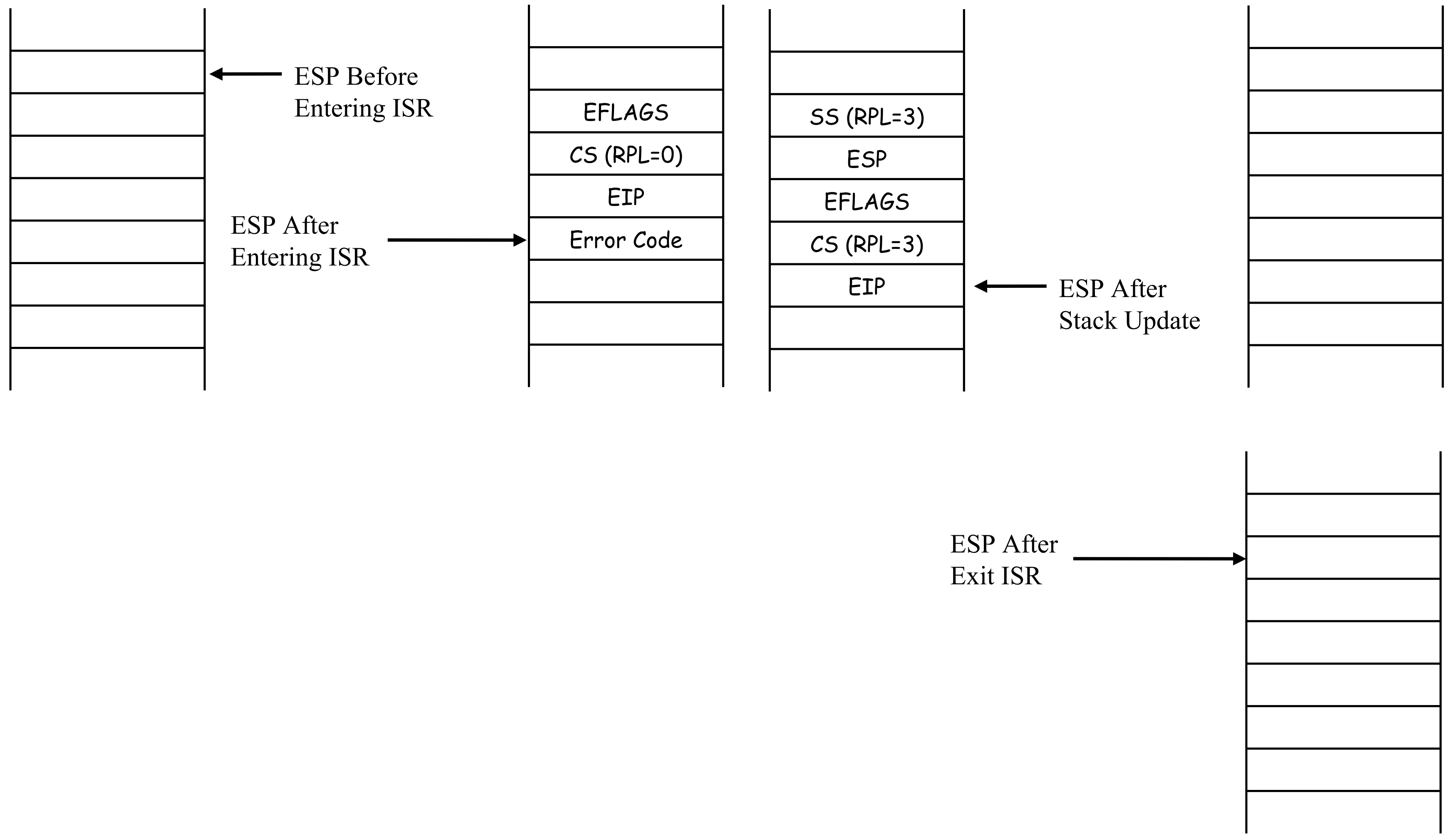
x86 特权级 –切换特权级 (0 to 3)



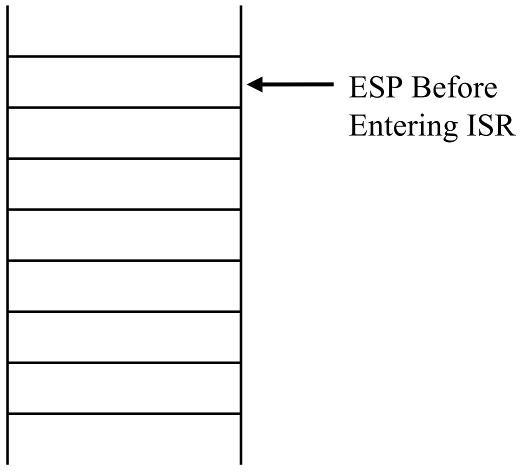
x86 特权级 –切换特权级 (0 to 3)



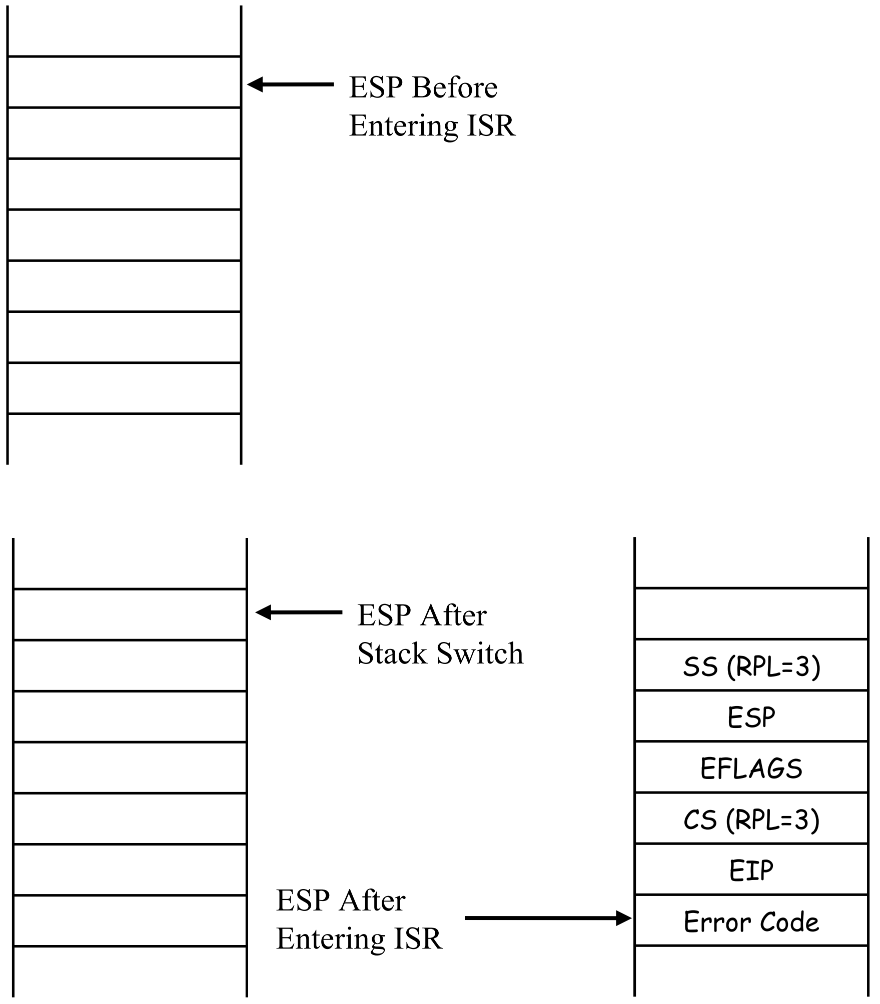
x86 特权级 –切换特权级 (0 to 3)



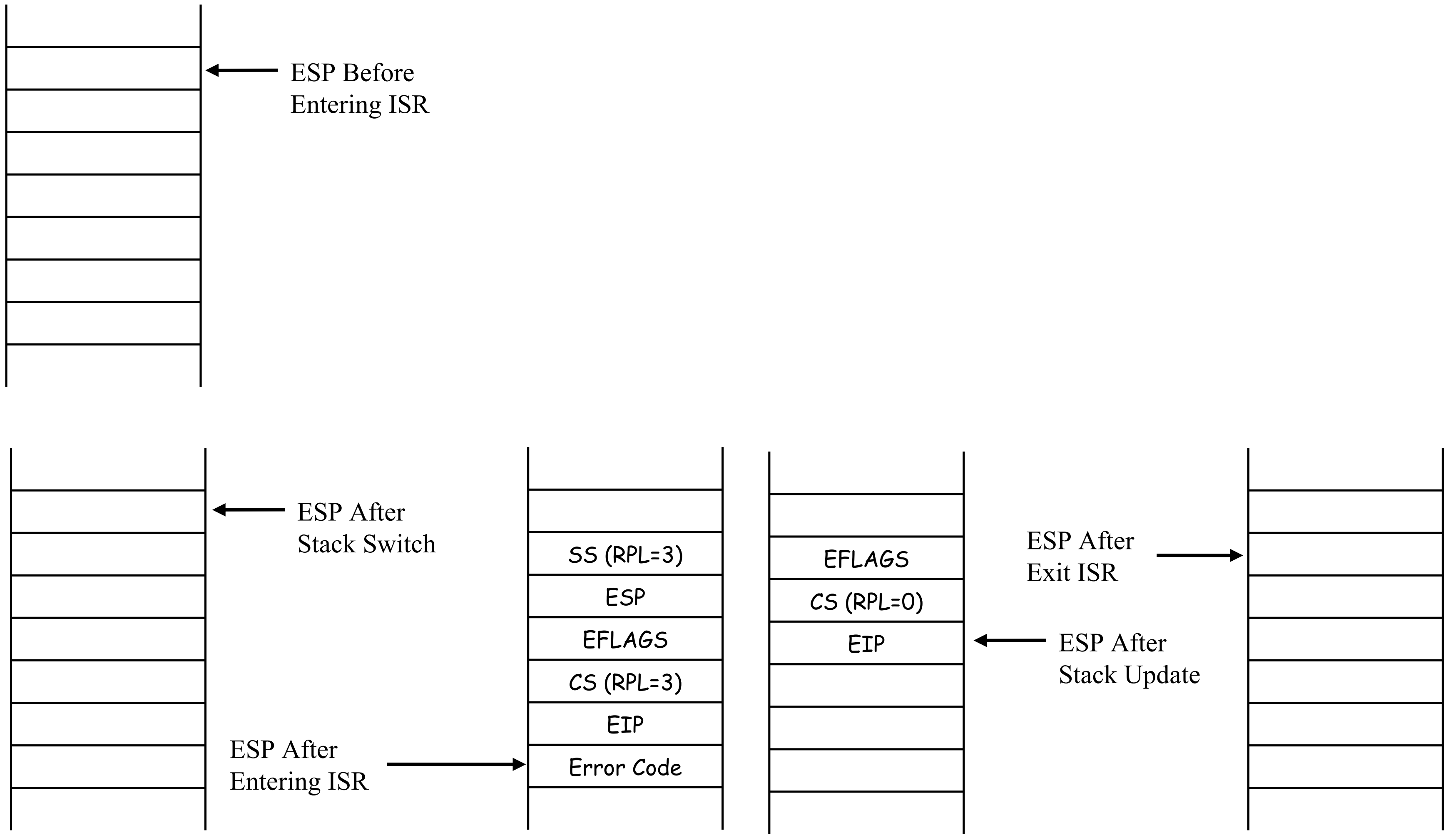
x86 特权级 –切换特权级(3 to 0)



x86 特权级 –切换特权级(3 to 0)

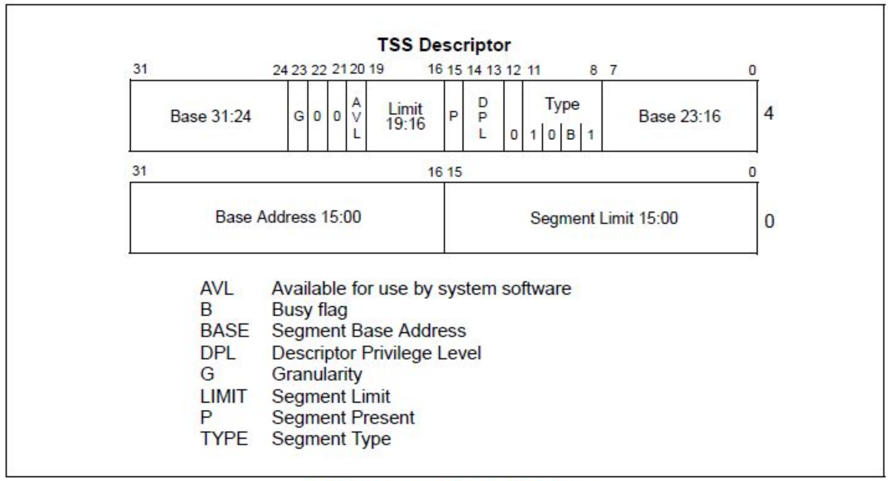


x86 特权级 –切换特权级(3 to 0)



x86 特权级 – 栈切换中获取新的栈

- TSS = Task State **Segment**

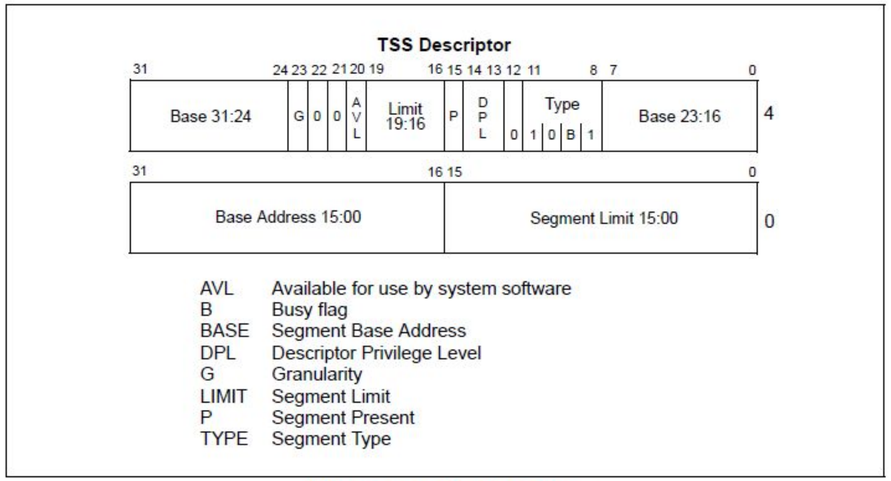


AVL	Available for use by system software
B	Busy flag
BASE	Segment Base Address
DPL	Descriptor Privilege Level
G	Granularity
LIMIT	Segment Limit
P	Segment Present
TYPE	Segment Type

Figure 7-3. TSS Descriptor

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

Figure 7-3. TSS Descriptor

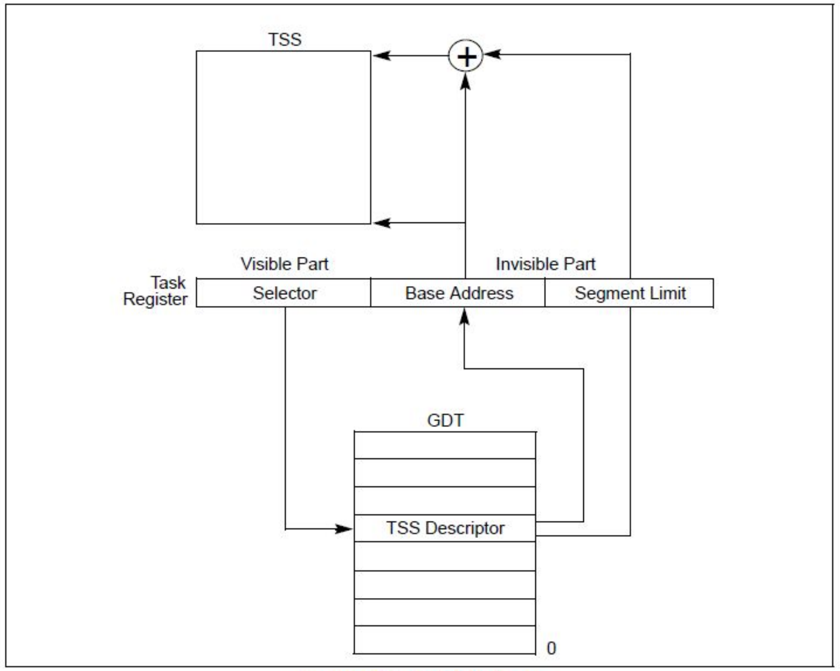
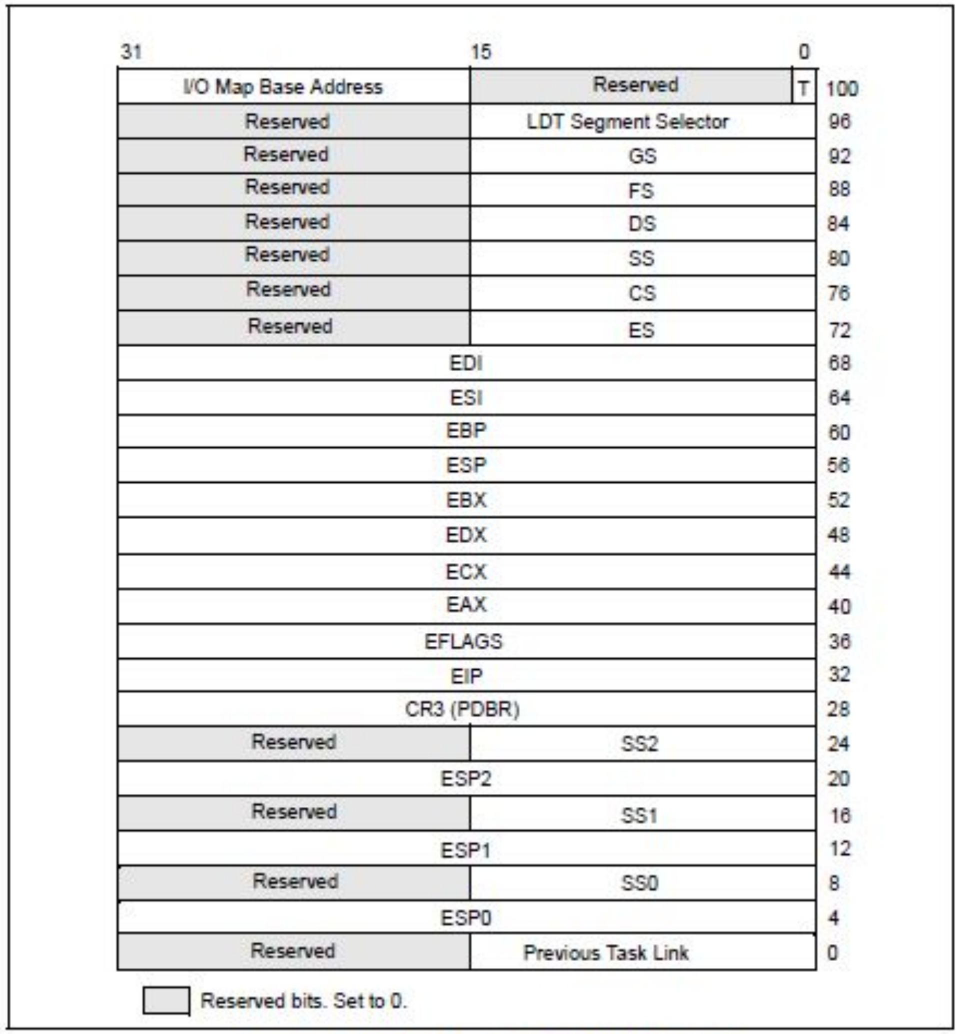


Figure 7-5. Task Register

x86 特权级 – TSS 格式

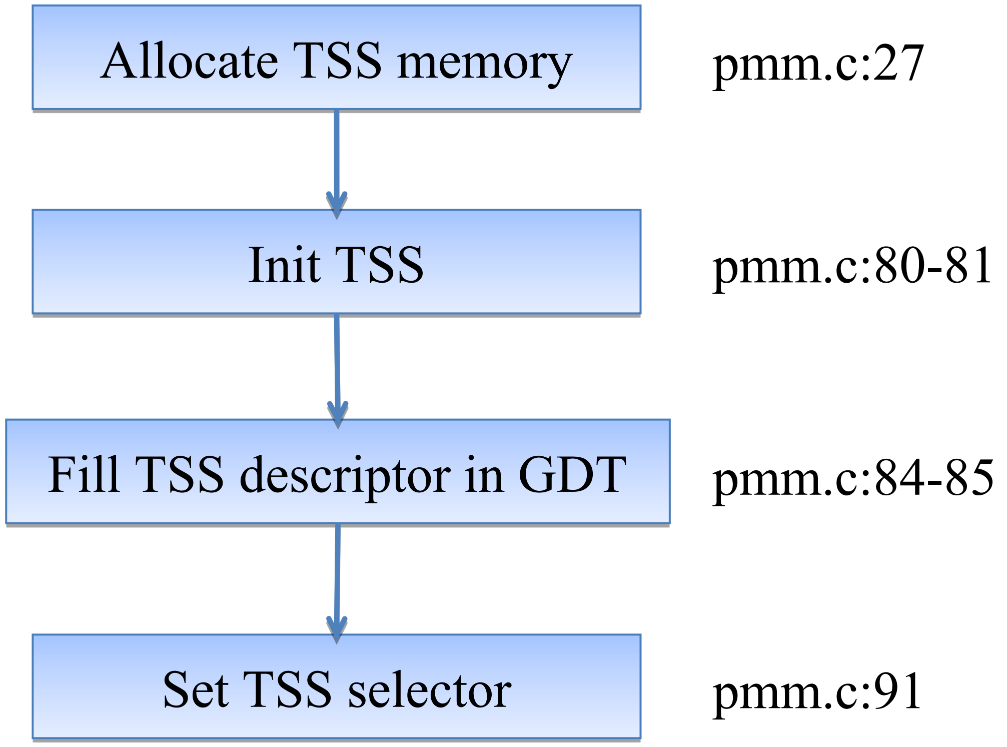


31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24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

▨ Reserved bits. Set to 0.

Figure 7-2. 32-Bit Task-State Segment (TSS)

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

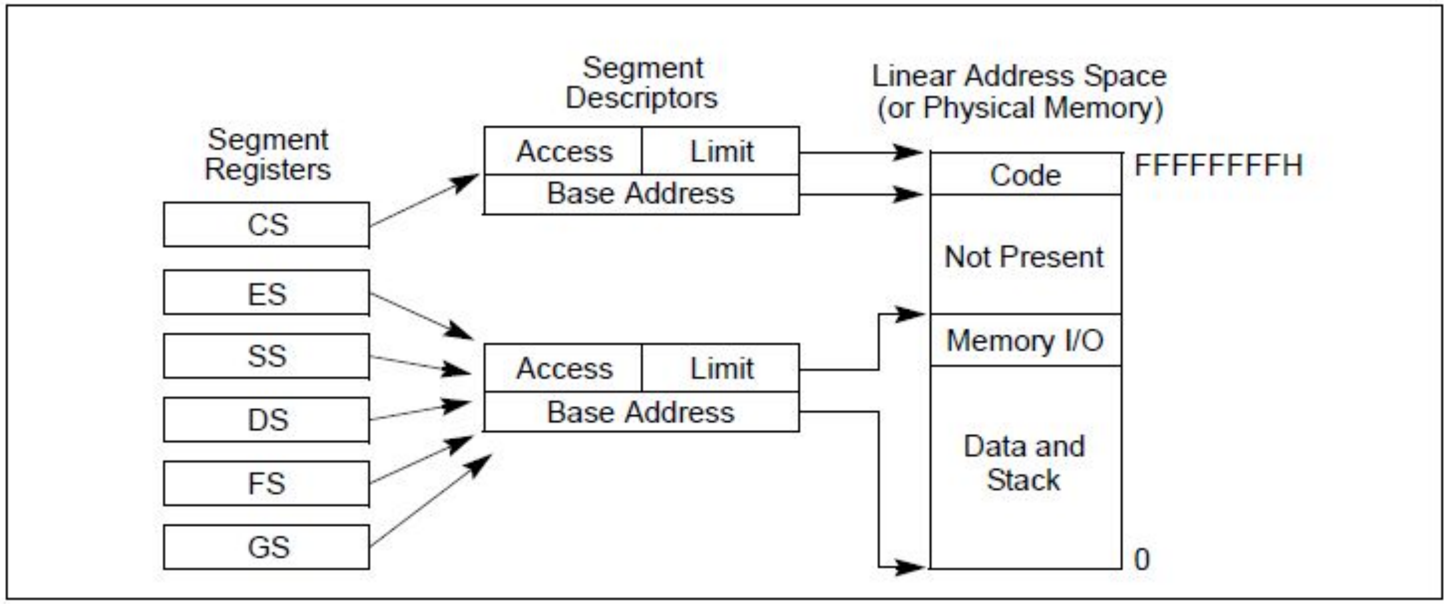


Figure 3-3. Protected Flat Model

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

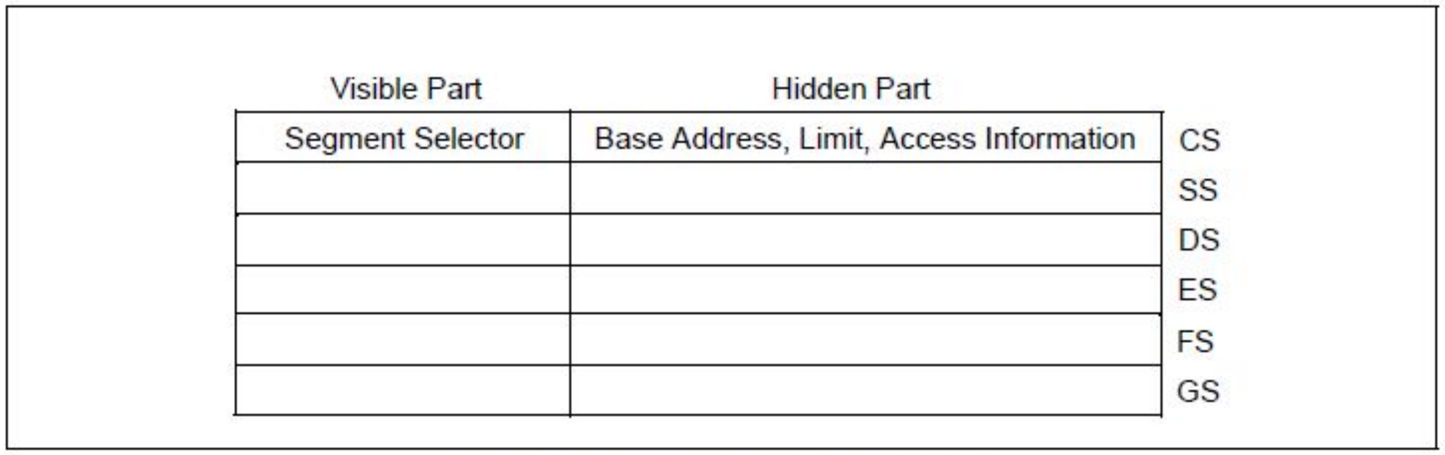
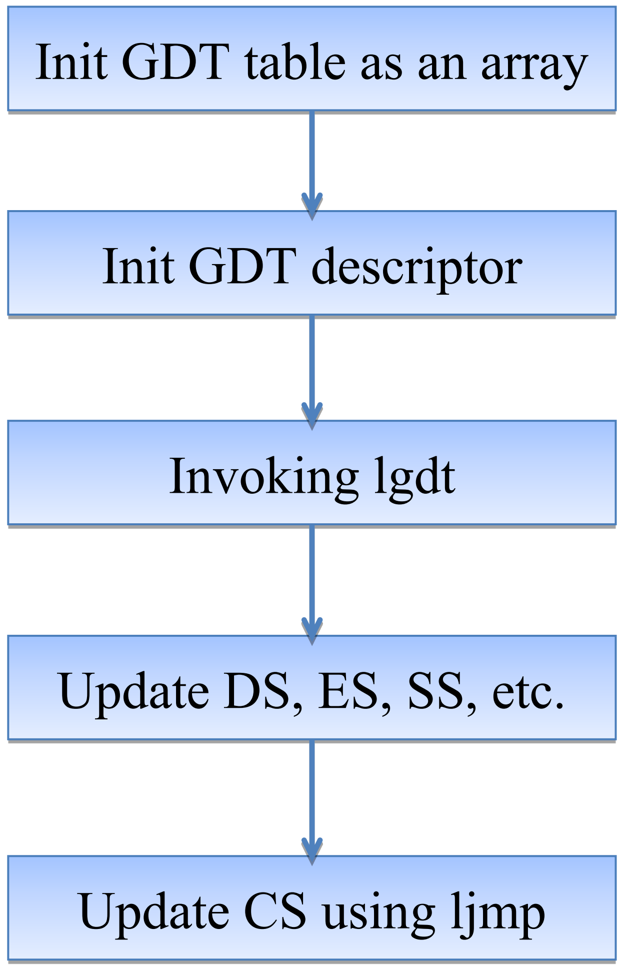


Figure 3-7. Segment Registers

- 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)



entry.S:42 (base address is -0xC0000000)

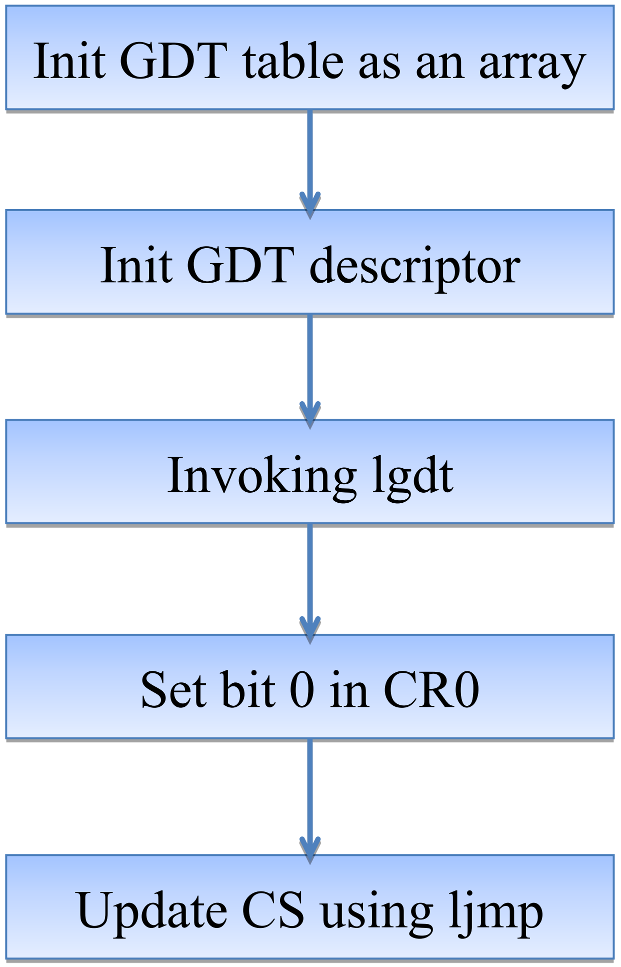
entry.S:46

entry.S:11

entry.S:12-15

entry.S:17

x86 MMU – 建立 GDT tables (bootloader)



bootasm.S:100 (base address is 0x0)

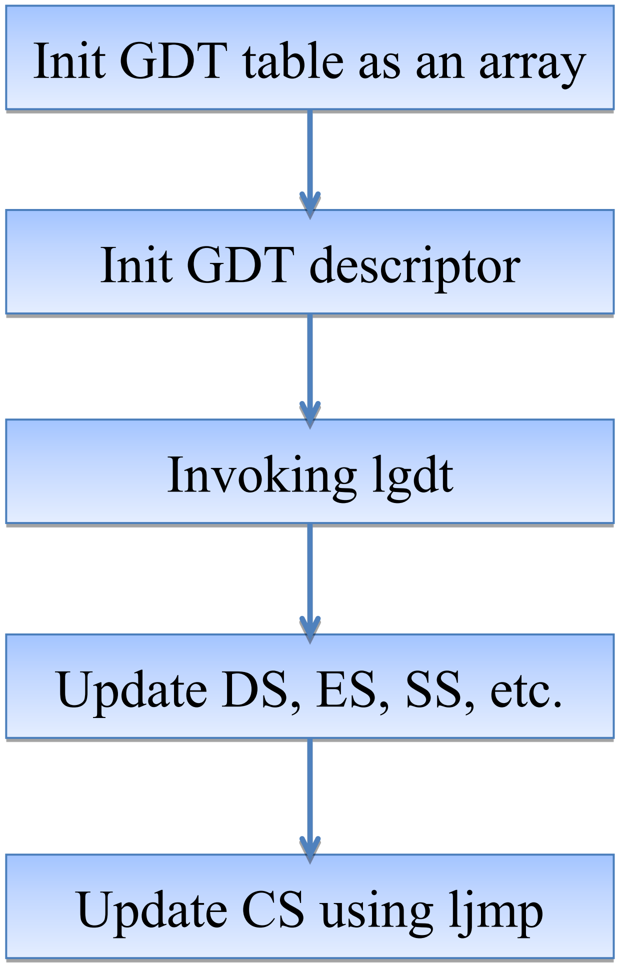
bootasm.S:105

bootasm.S:69

bootasm.S:70-72

bootasm.S:76

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



pmm.c:76 (base address is 0x0)

pmm.c:85

pmm.c:99

pmm.c:100-104

pmm.c:106

Why bothering?!

x86 MMU – 页机制概述

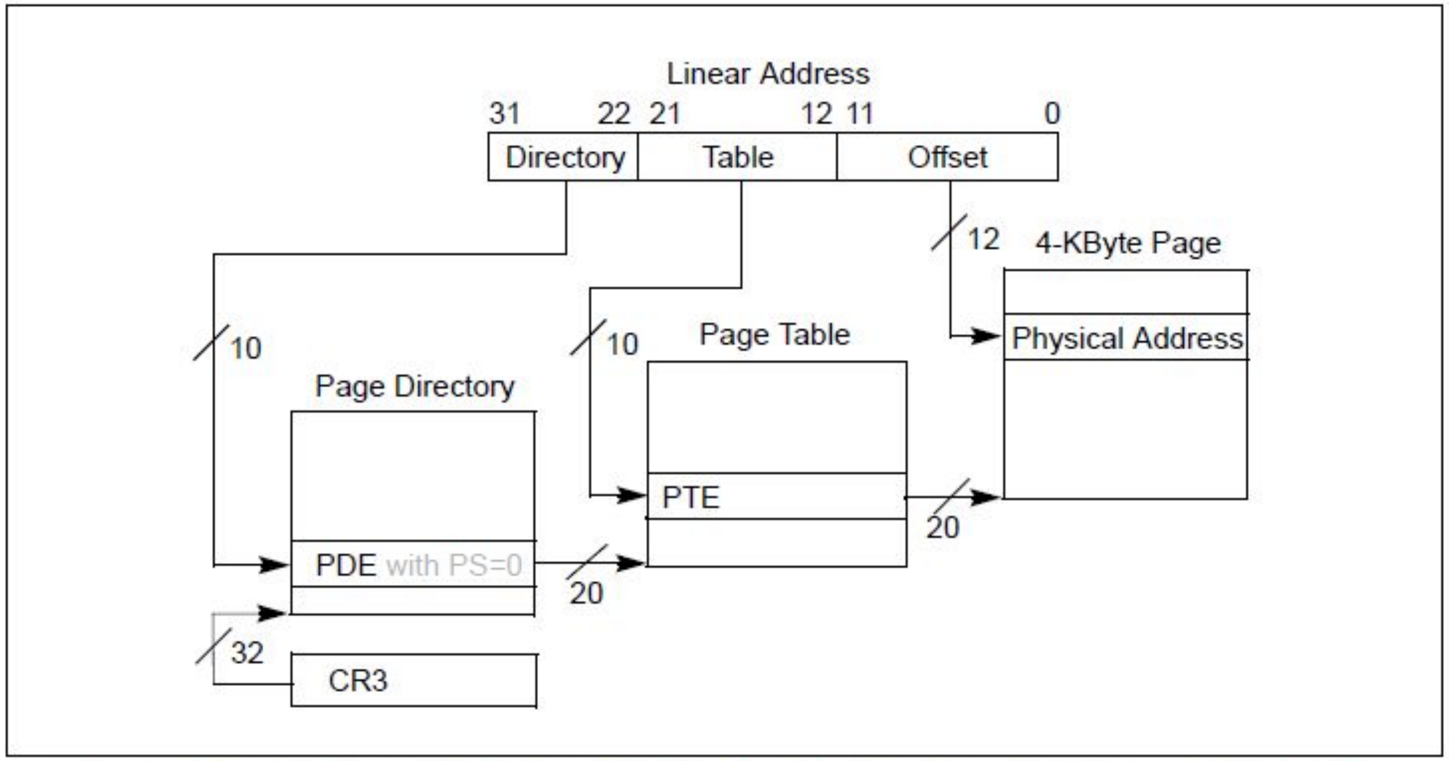
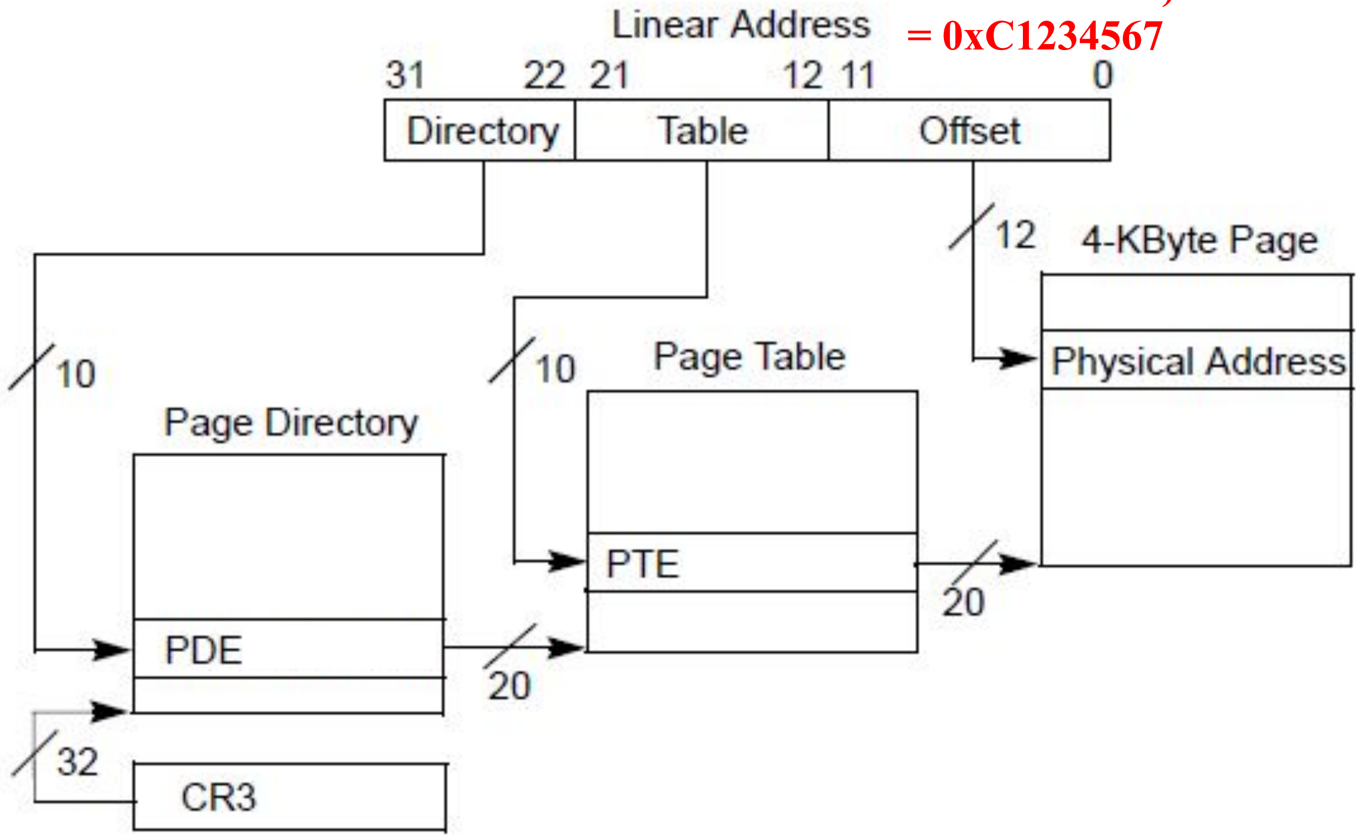


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

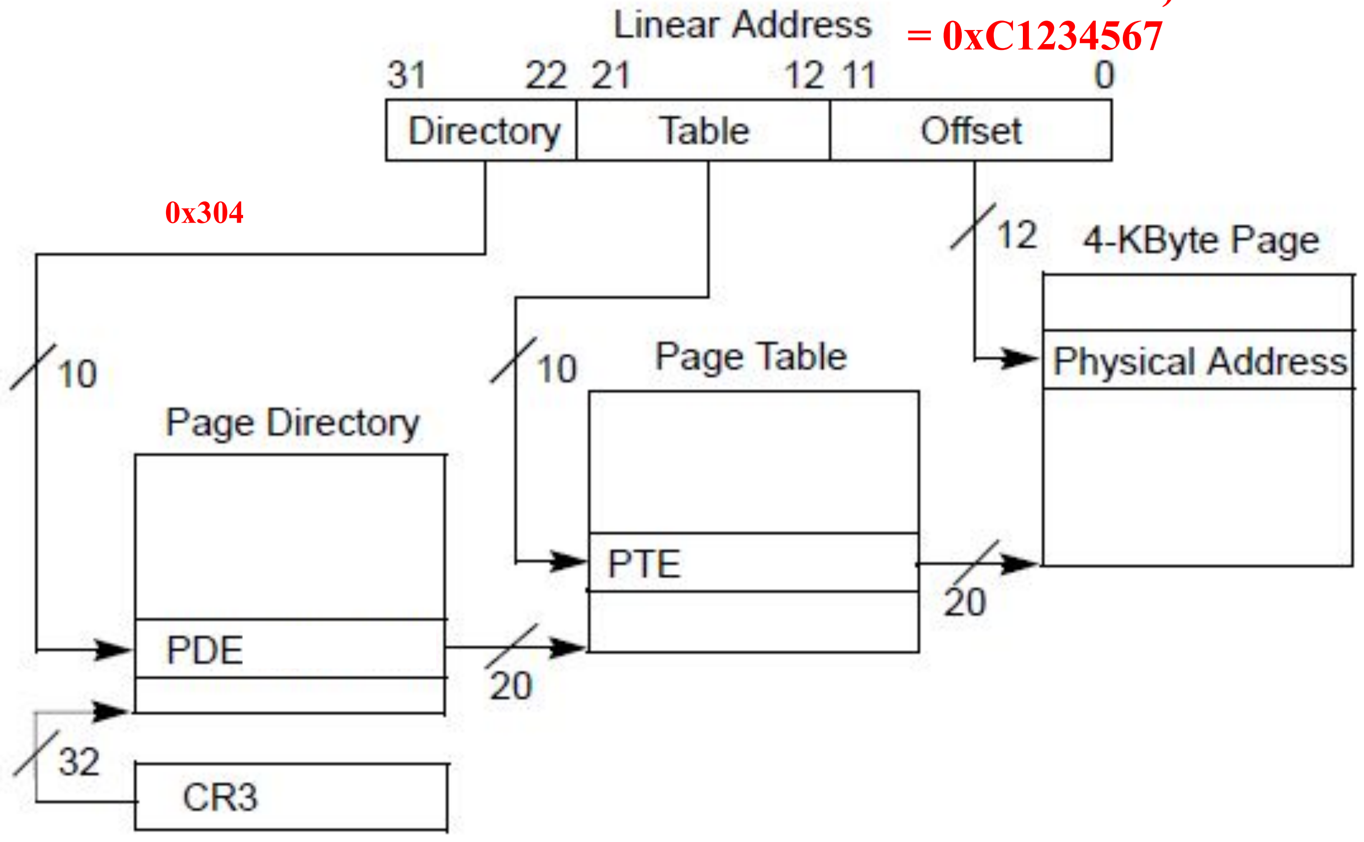
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567

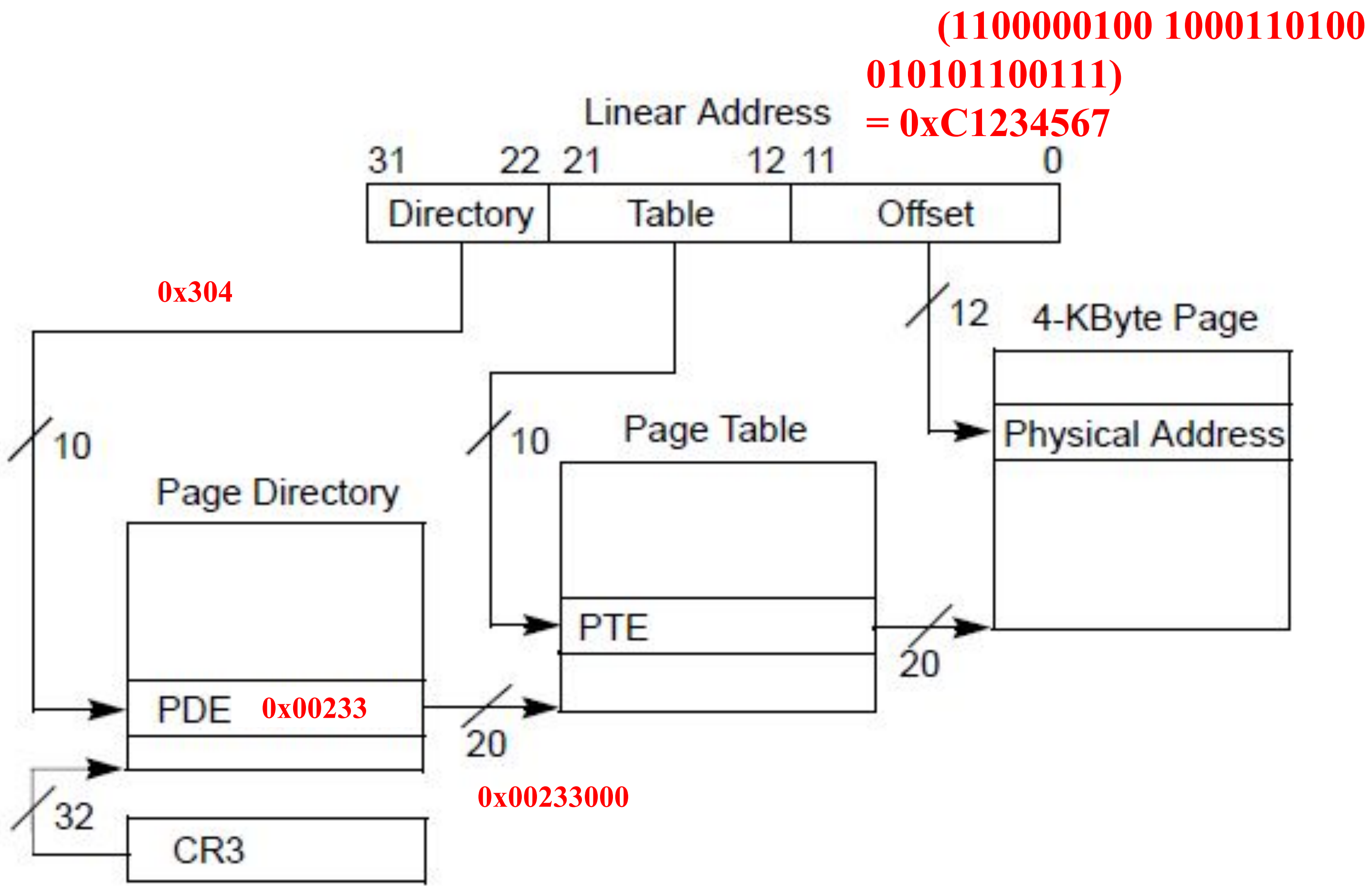


x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567

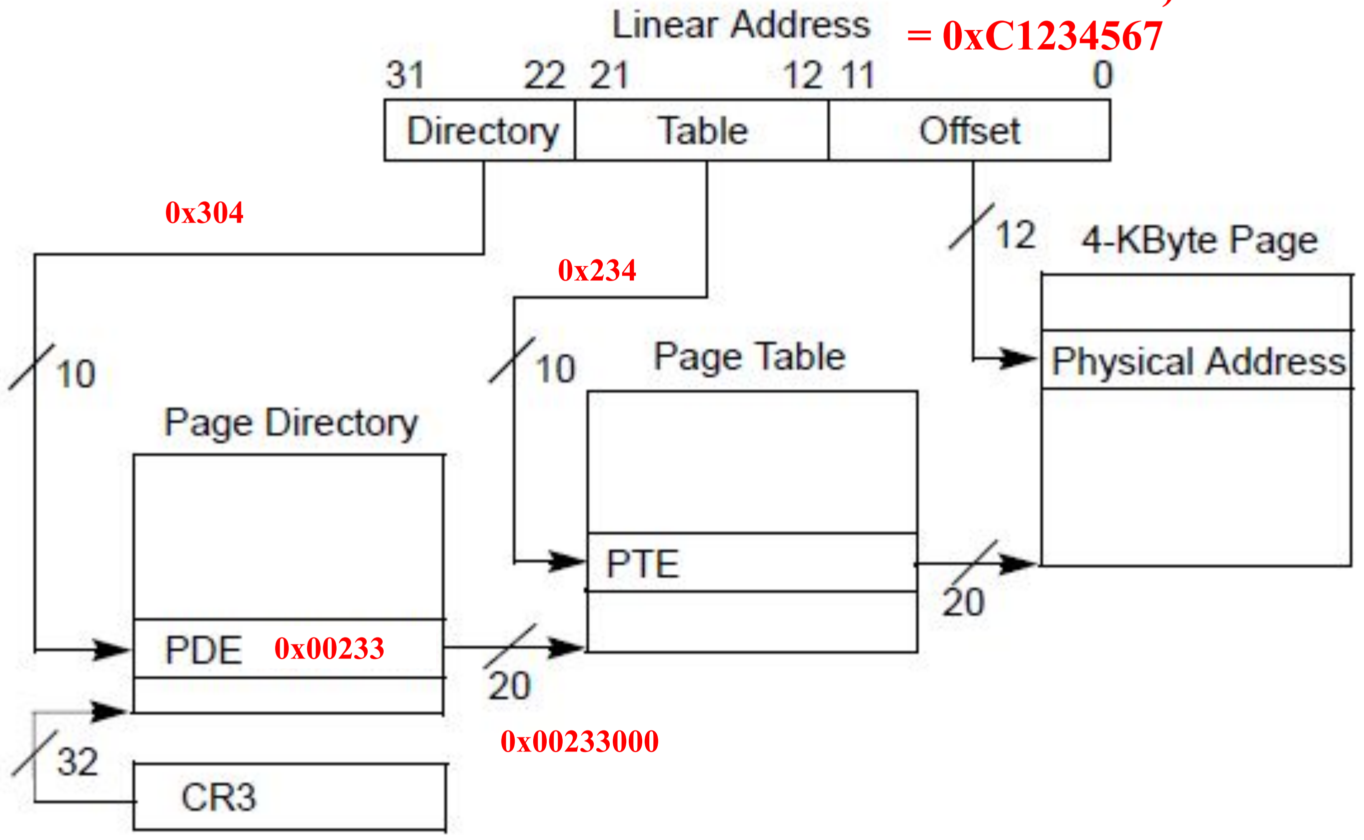


x86 MMU – 页机制举例

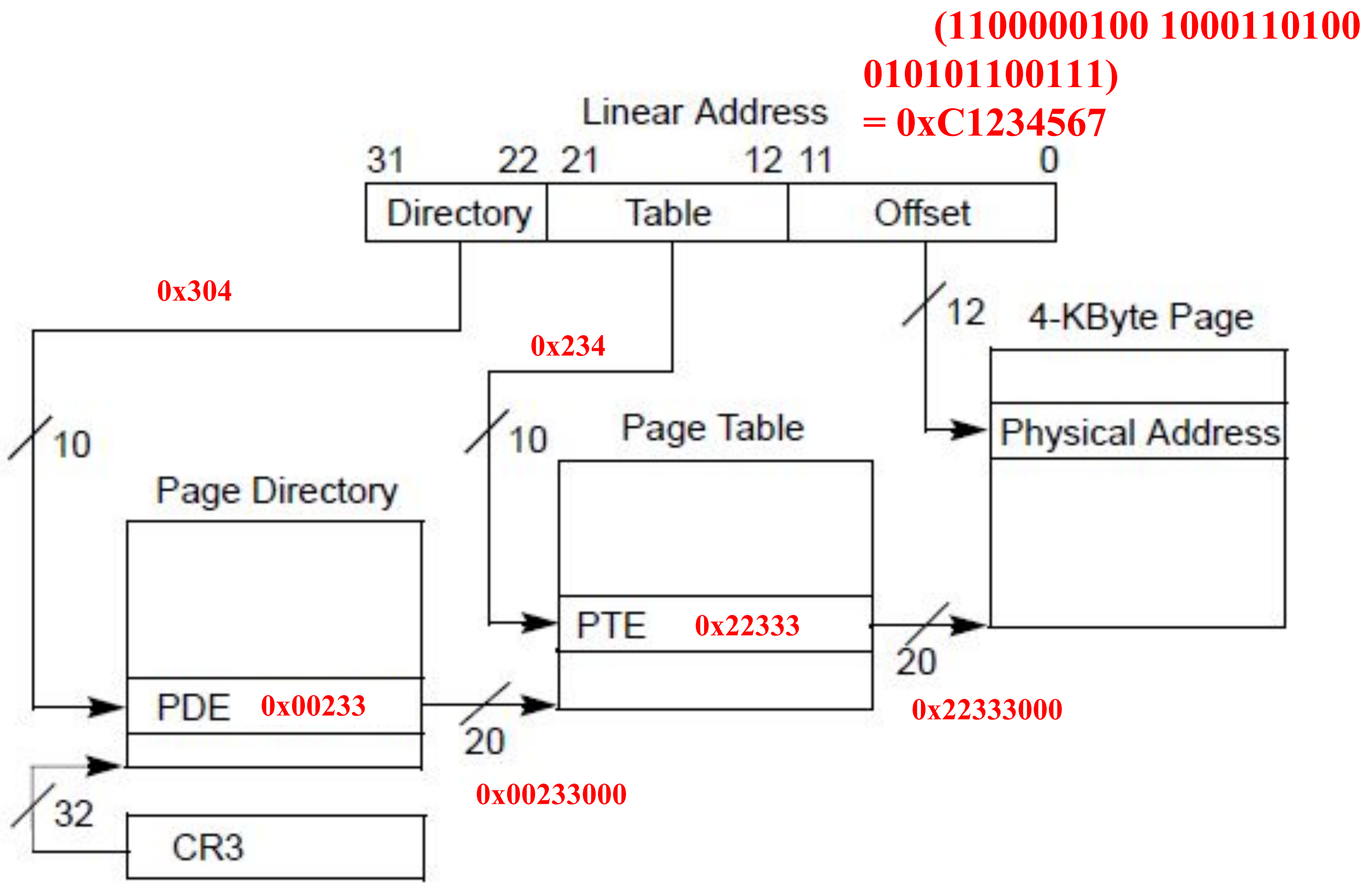


x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567

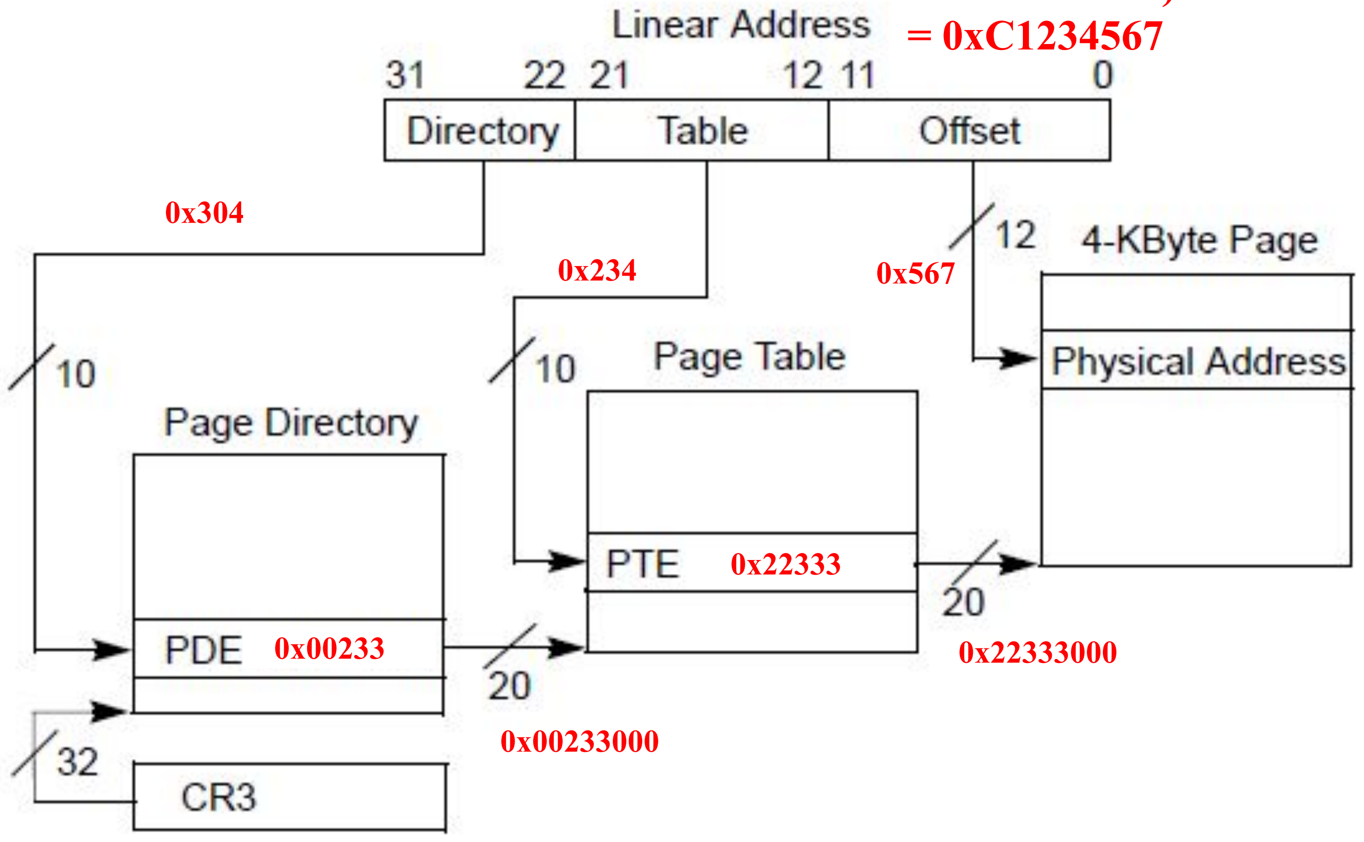


x86 MMU – 页机制举例



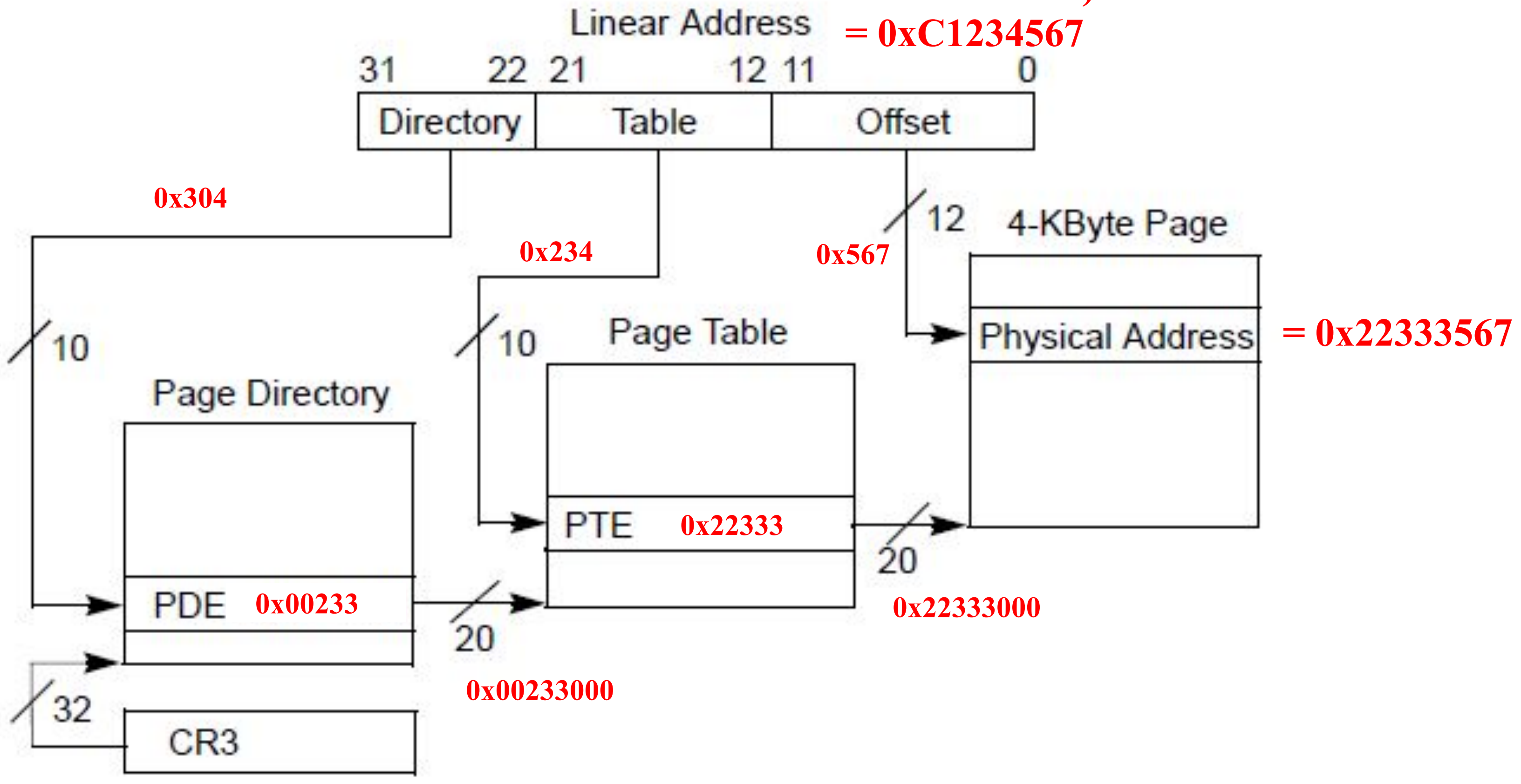
x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567

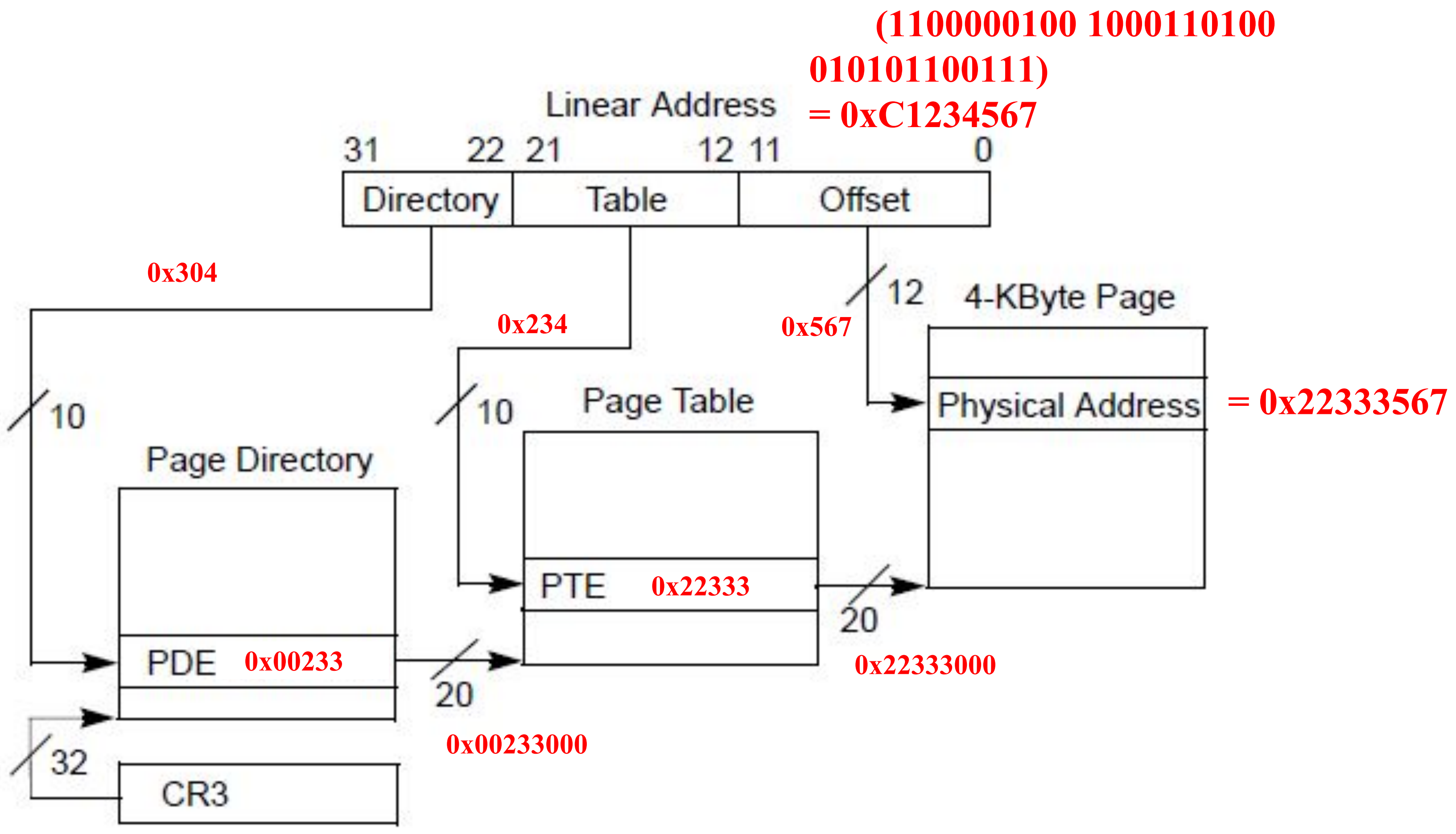


x86 MMU – 页机制举例

(1100000100 1000110100 010101100111)
= 0xC1234567

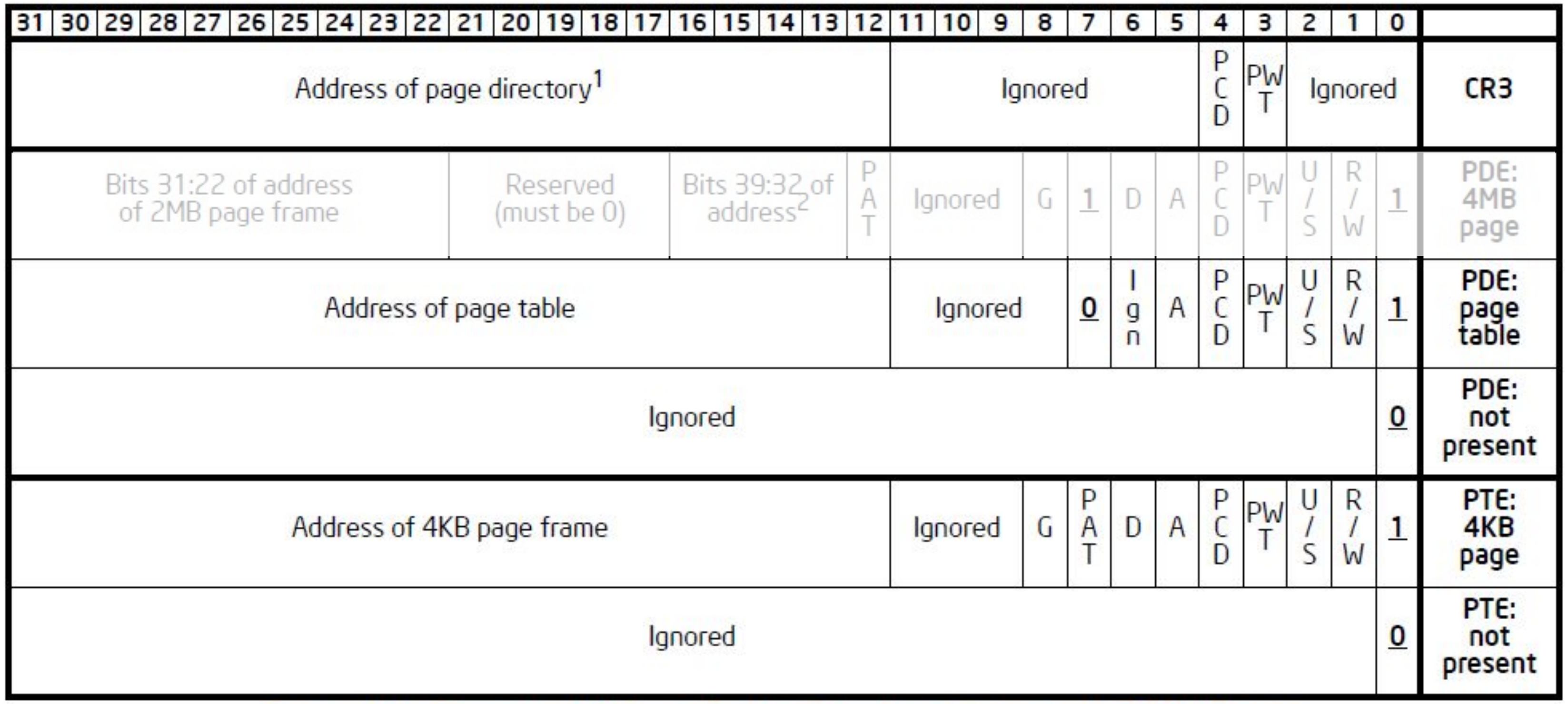


x86 MMU – 页机制举例



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							PCD	PWT	Ignored			CR3
Bits 31:22 of address of 2MB page frame										Reserved (must be 0)					Bits 39:32 of address[2]				PAT	Ignored			G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 4MB page
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored																															0	PDE: not present
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored																															0	PTE: not present

Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

- 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
- D: 1 if this page has been written
- You may ignore others for now

x86 MMU – 使能页机制（enable paging）

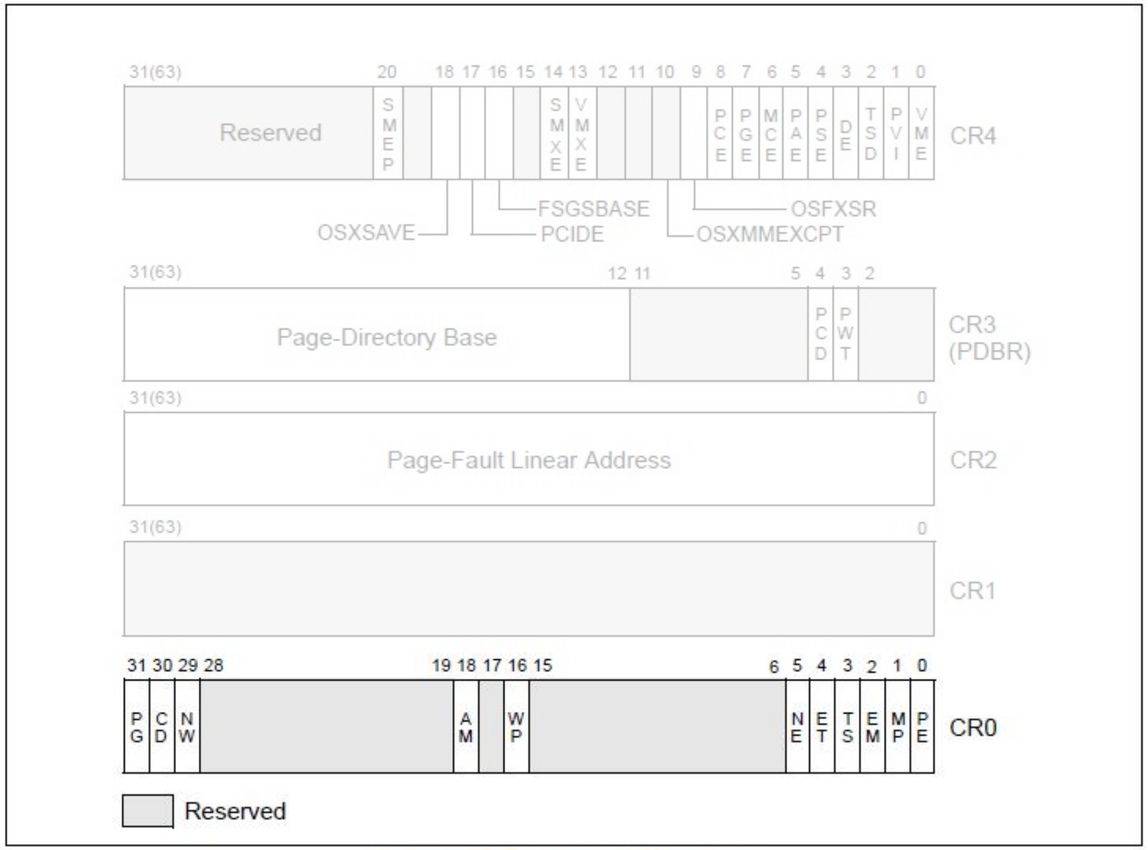
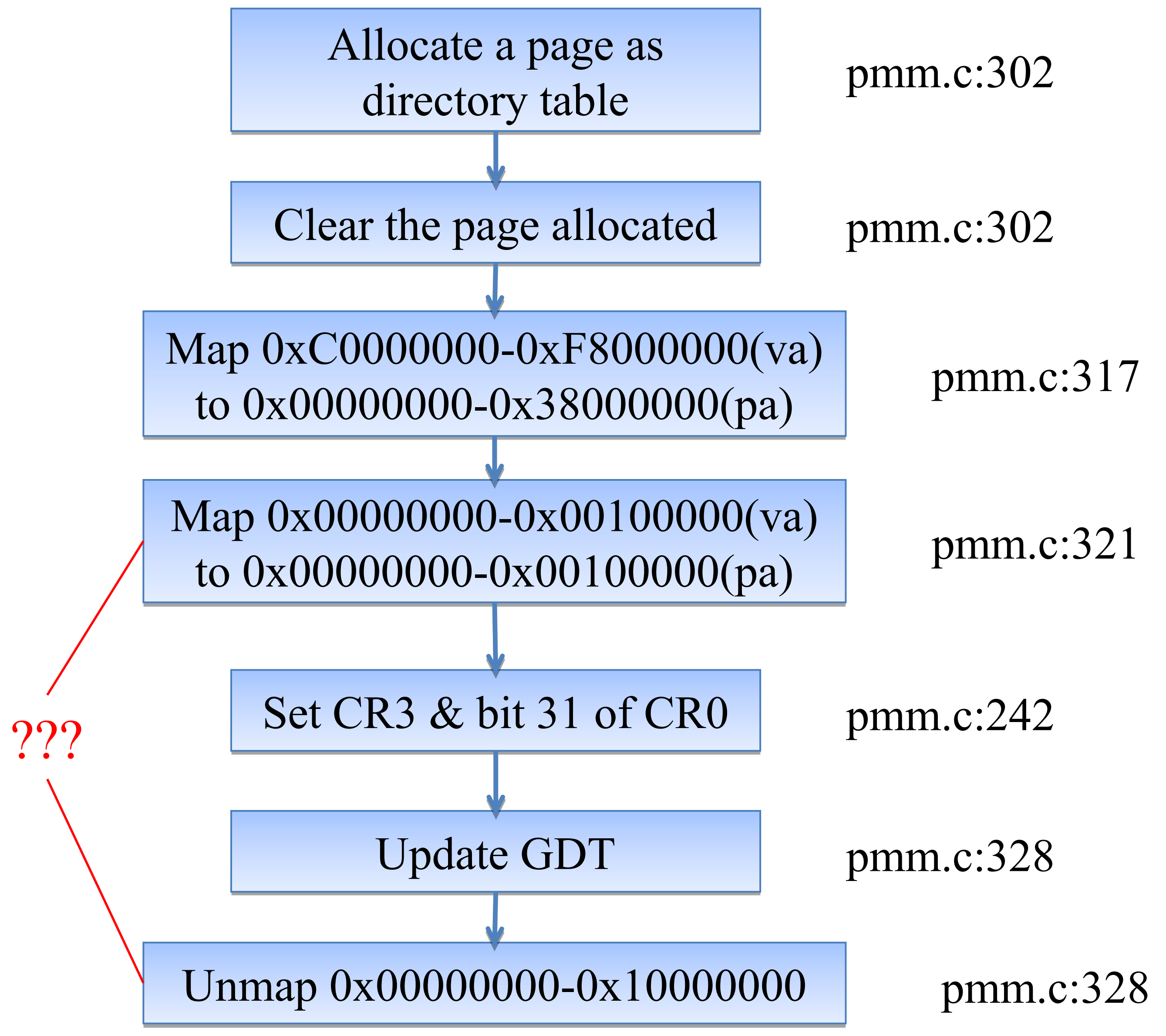


Figure 2-7. Control Registers

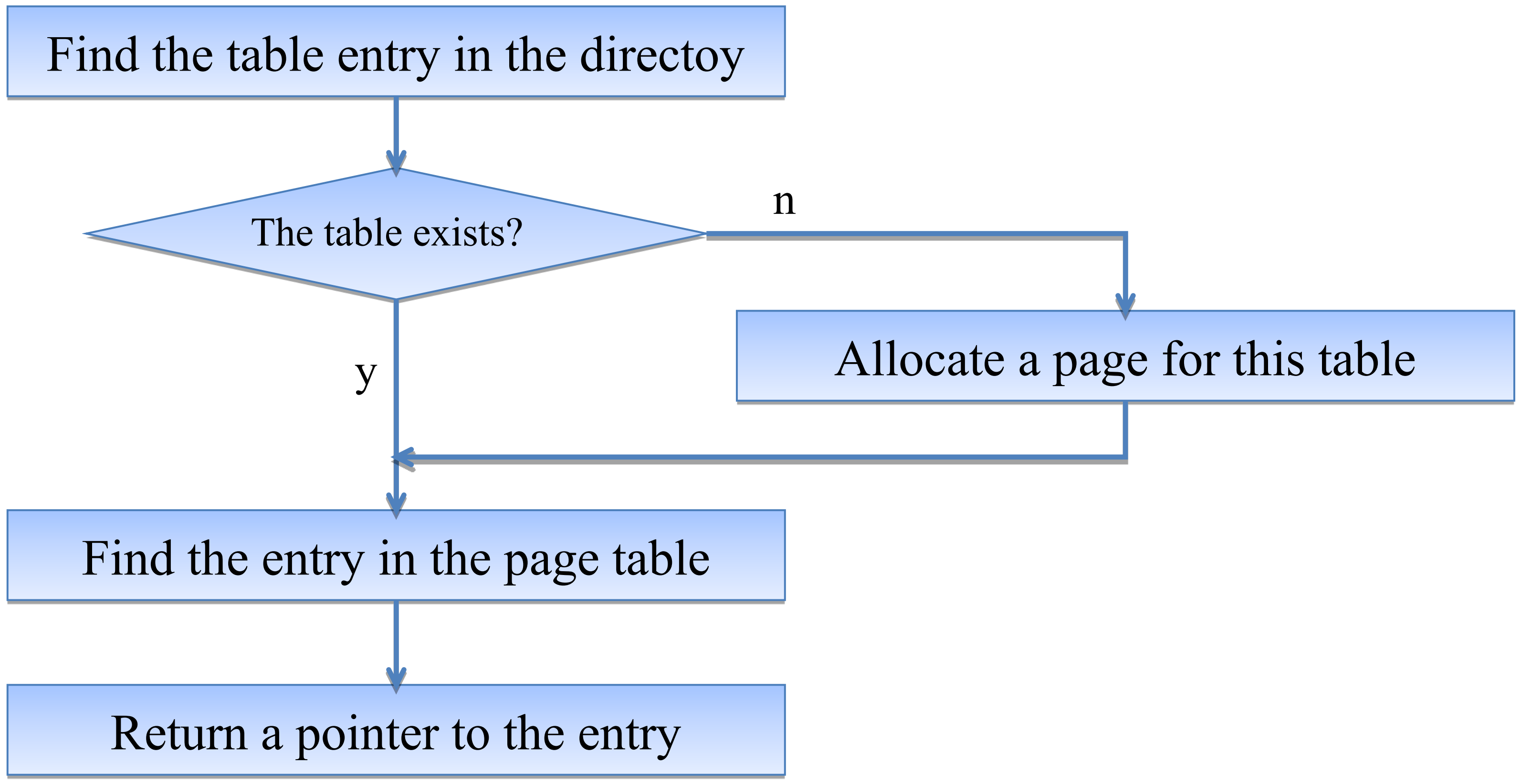
- 为了在保护模式下使能页机制, OS需要置CR0寄存器中的 bit 31 (PG)

x86 MMU – 建立页表（page tables）



Allocate a page as directory table
pmm.c:302
Clear the page allocated
pmm.c:302
Map 0xC0000000-0xF8000000(va) to 0x00000000-0x38000000(pa)
pmm.c:317
Map 0x00000000-0x00100000(va) to 0x00000000-0x00100000(pa)
pmm.c:321
Set CR3 & bit 31 of CR0
pmm.c:242
Update GDT
pmm.c:328
Unmap 0x00000000-0x10000000
pmm.c:328
???

x86 MMU – 在页表中建立页的映射关系



Find the table entry in the directoy
The table exists?
n
y
Allocate a page for this table
Find the entry in the page table
Return a pointer to the entry

YOUR WORK!

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

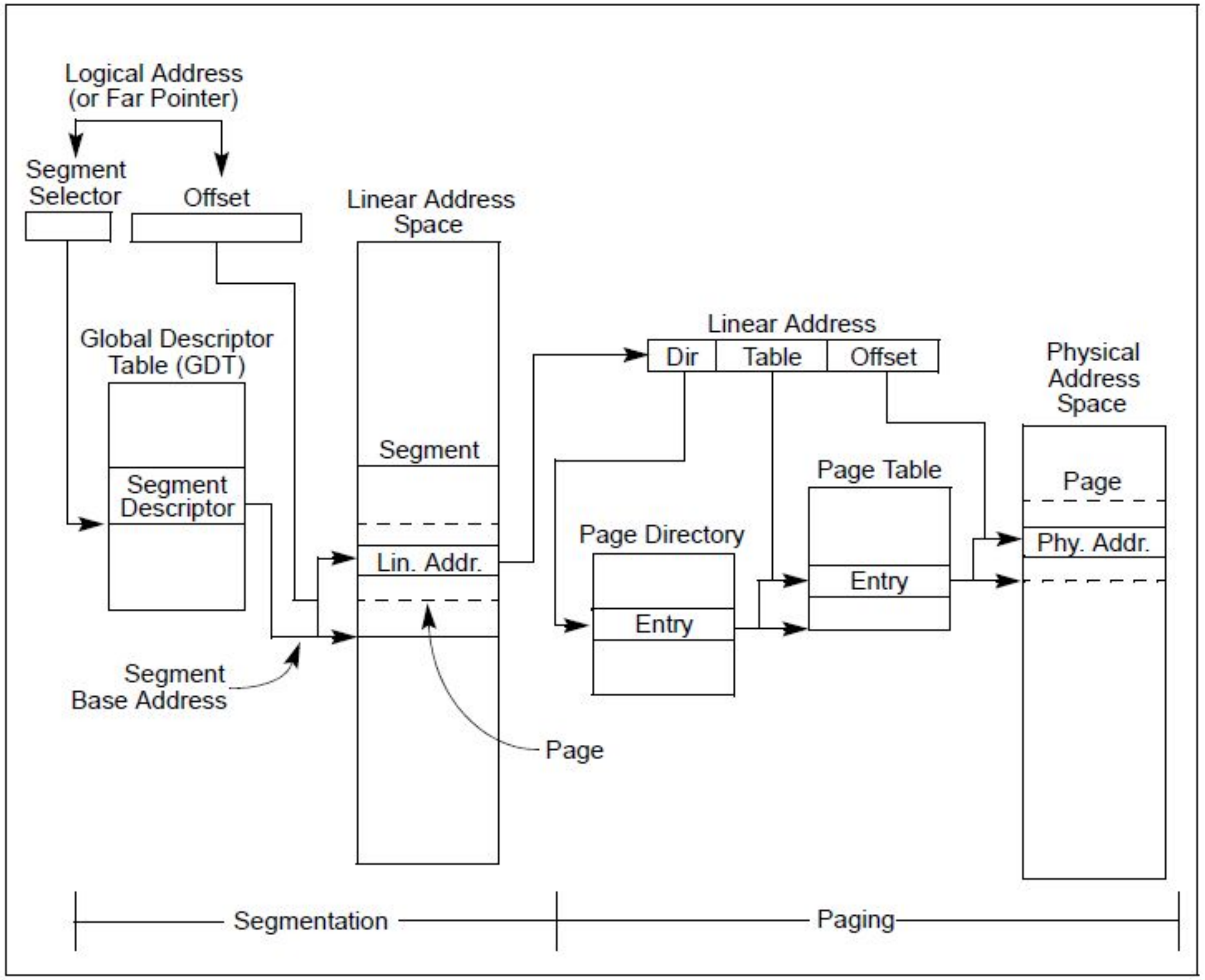
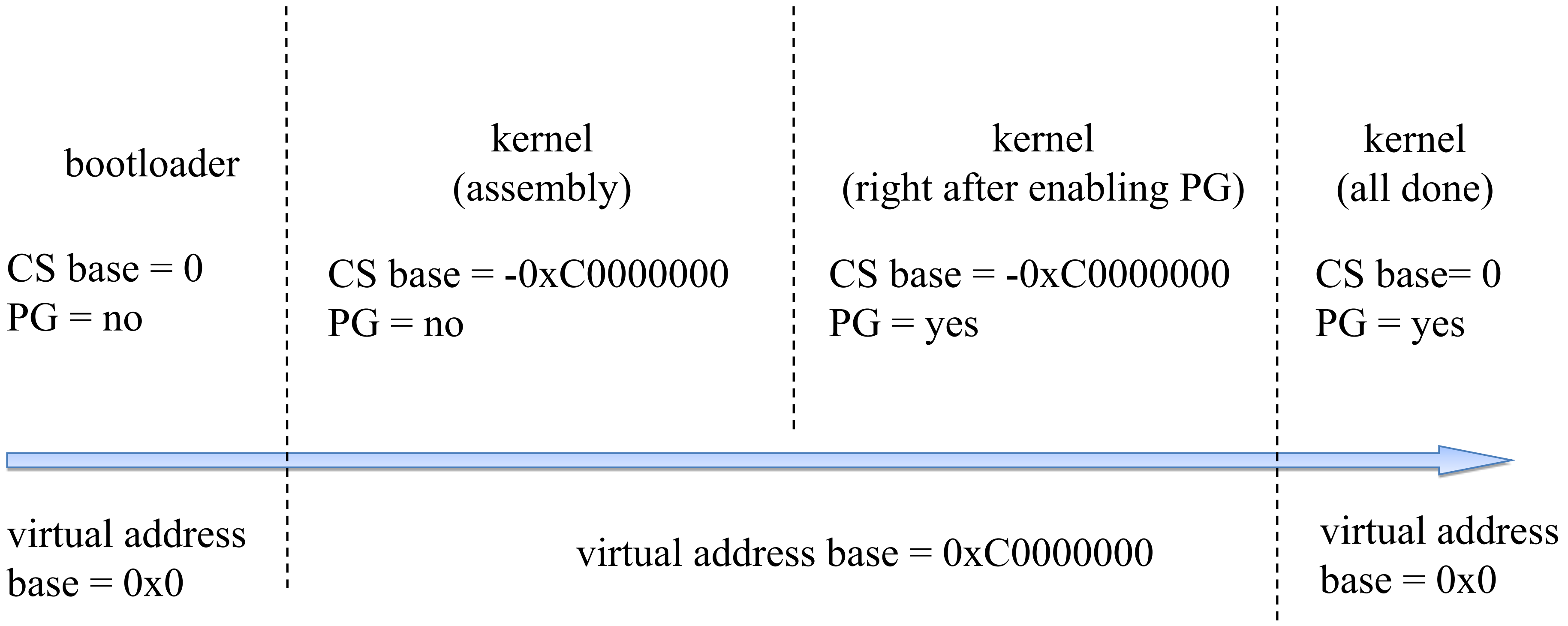


Figure 3-1. Segmentation and Paging

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