# Operating Systems

## Introduction to Lab 2

Department of Computer Science & Technology
Tsinghua University

# Outline

- x86 privilege levels
- x86 hardware Memory Management Unit (MMU)

- Know what's different in different privilege levels
- Know how to check what privilege level we are currently in
- Know how to switch privilege levels

# X86 PRIVILEGE LEVELS

# x86 privilege levels – introduction

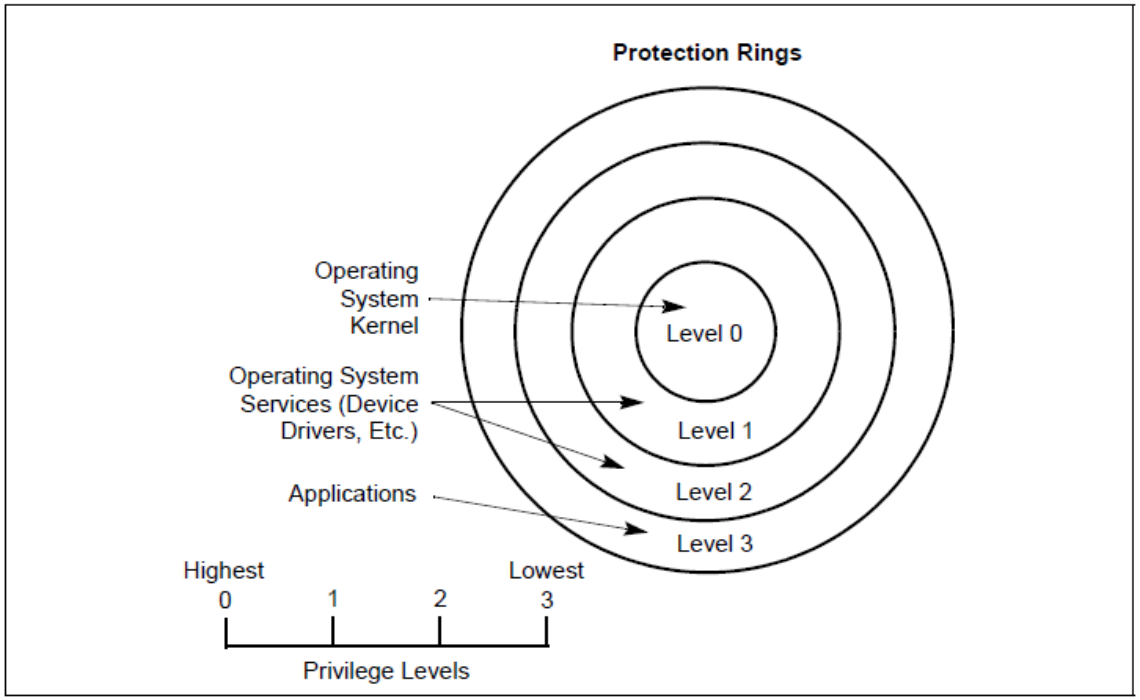



Figure 6-3. Protection Rings

- Linux and uCore uses ring 0 and ring 3 only.

# x86 privilege levels – what is different?

- Some instructions can only be executed in ring 0 (e.g. lgdt).
- Privilege levels will be checked when
  - accessing data segments
  - accessing pages
  - entering ISRs
  - ……
- What happens if the check fails?

General Protection Fault!

# x86 privilege levels – current privilege level?

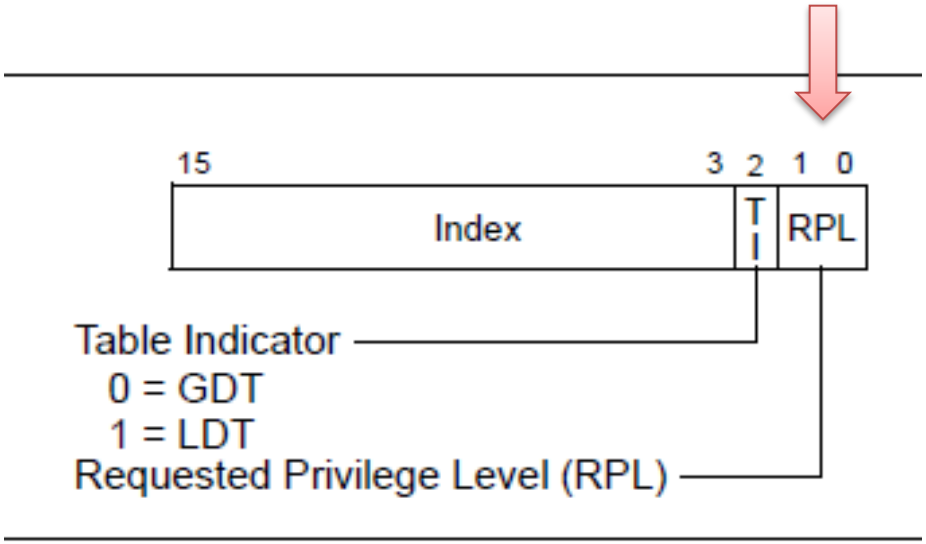



Figure 3-6. Segment Selector

# x86 privilege levels – privilege check example

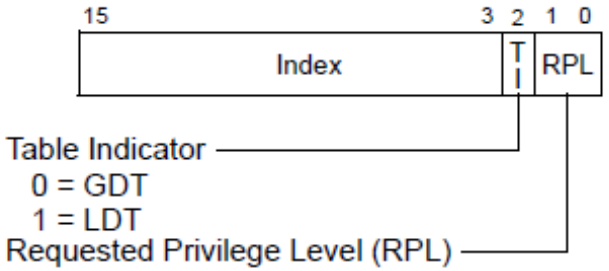
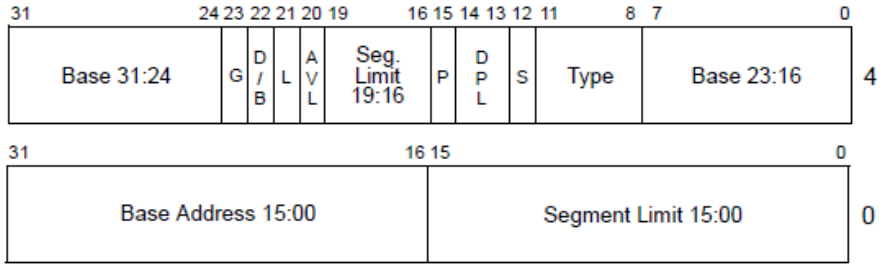
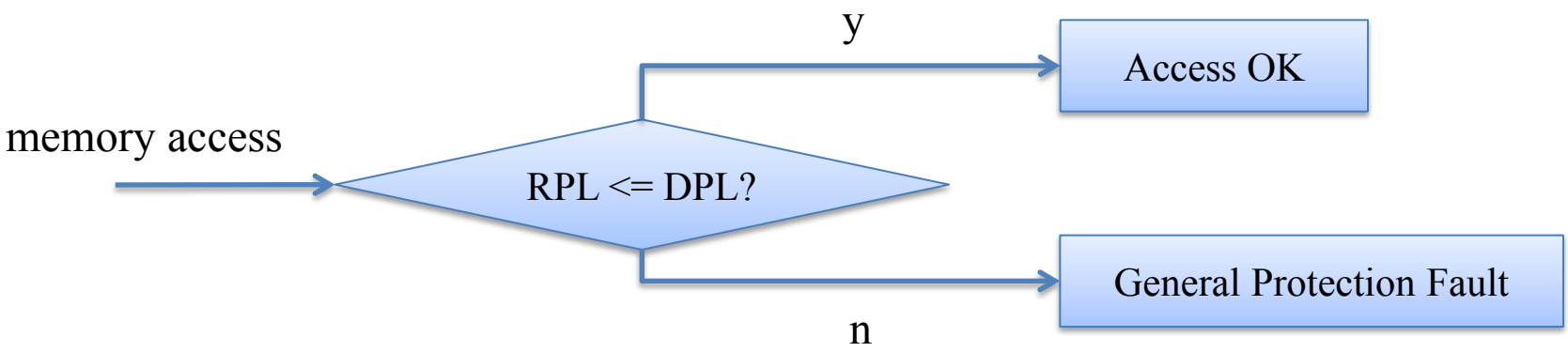



Figure 3-6. Segment Selector

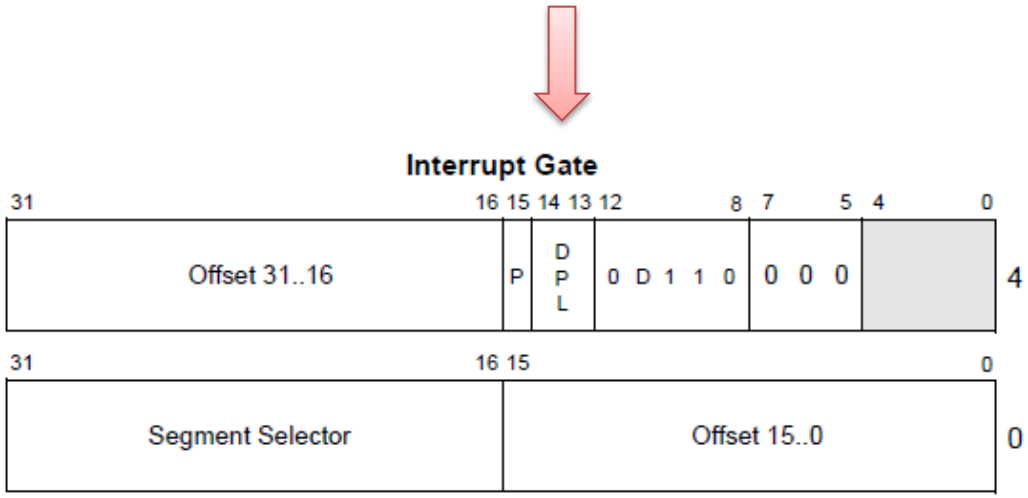



L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system;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

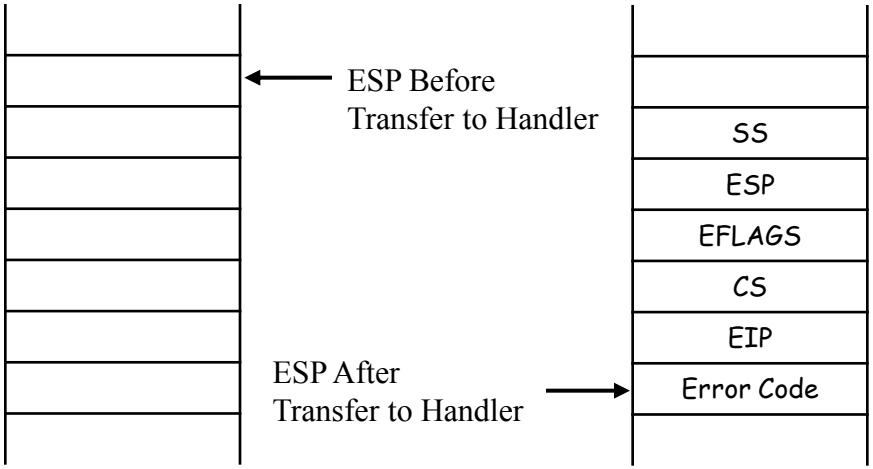
Figure 3-8. Segment Descriptor



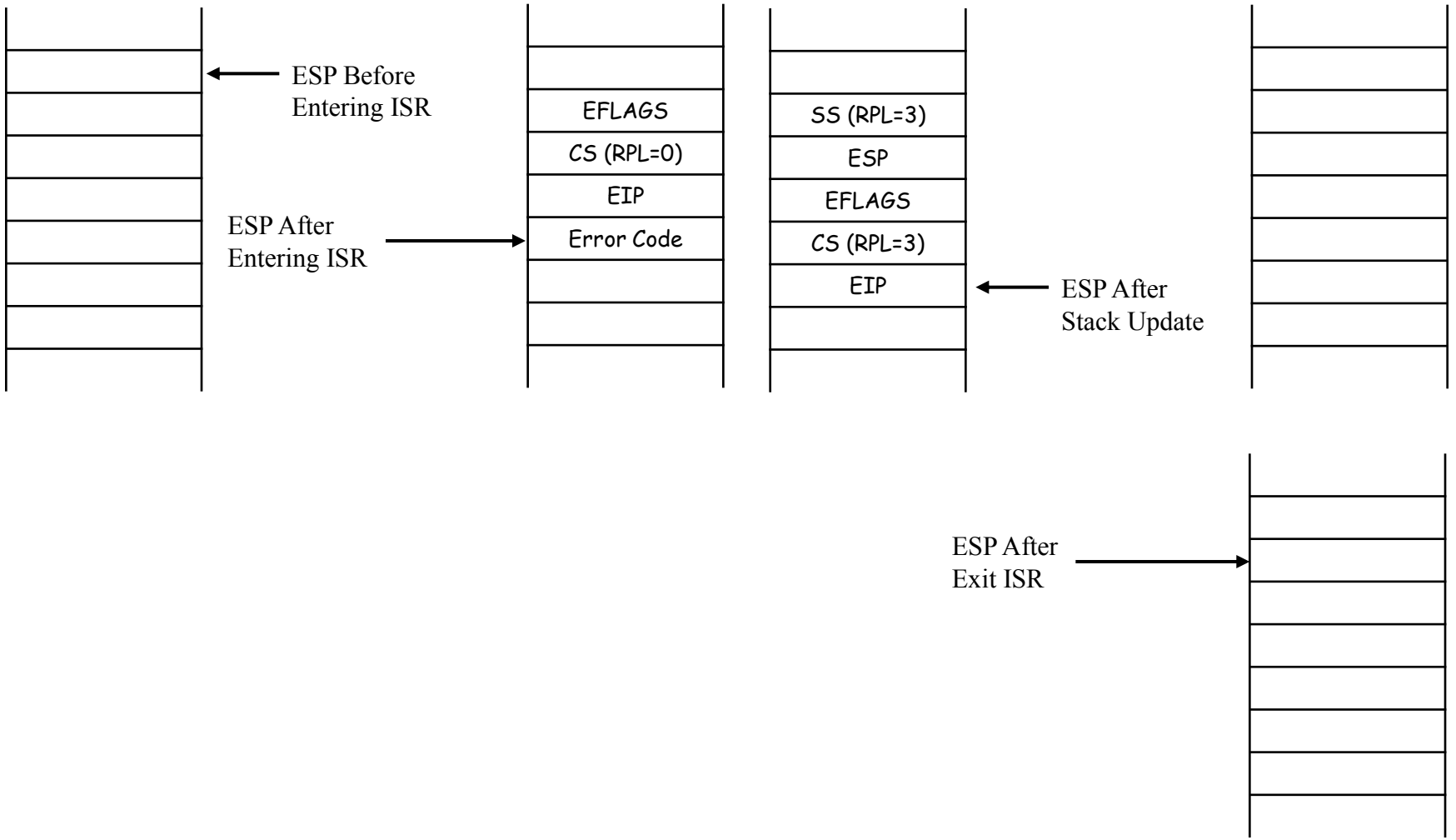
# x86 privilege levels – switch privilege levels by interrupt



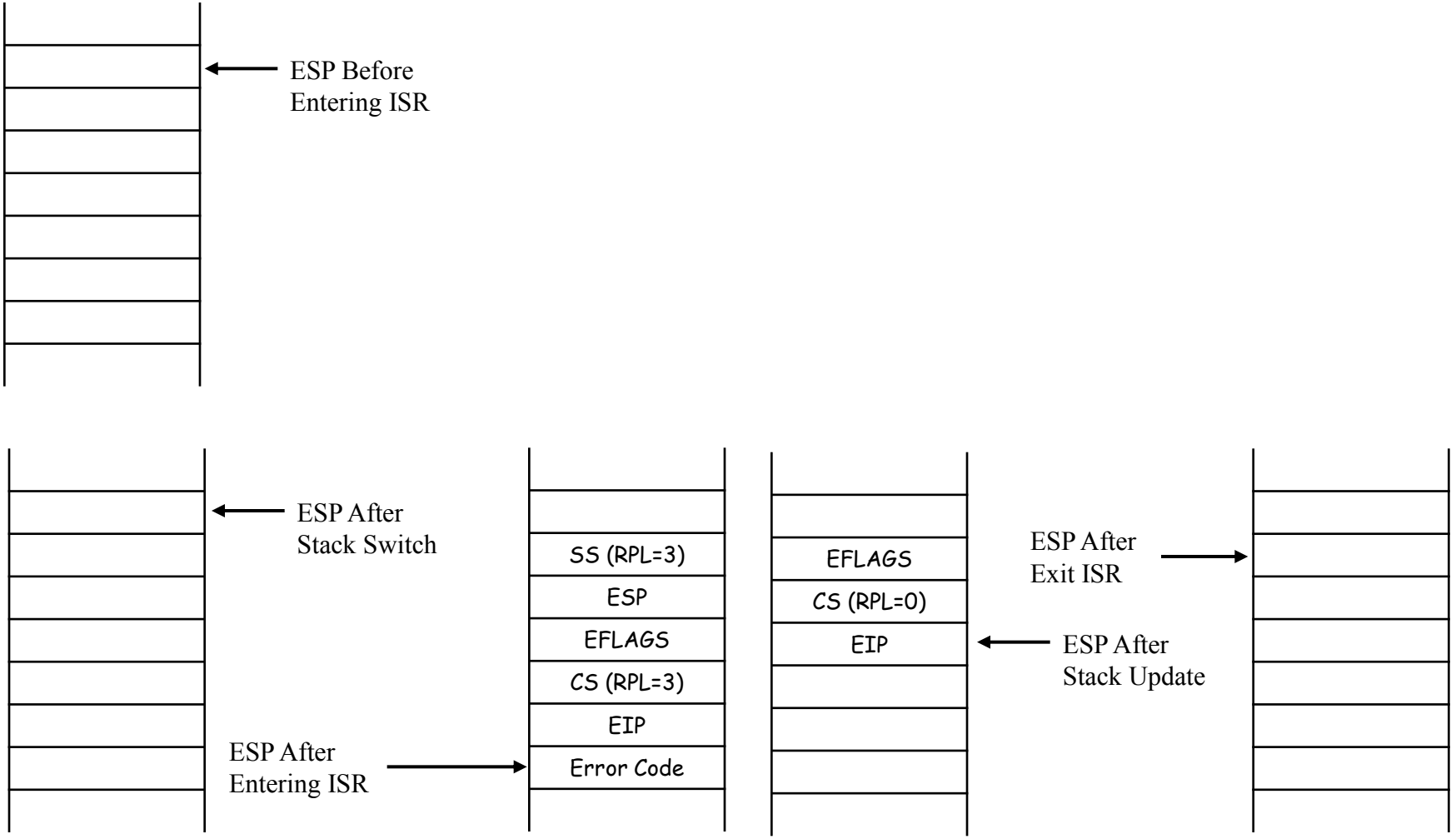

**Interrupt Stack Usage with Privilege-Level Change**



# x86 privilege levels – switch privilege levels (0 to 3)



# x86 privilege levels – switch privilege levels (3 to 0)



# x86 privilege levels – fetch new stack for stack switch

- TSS = Task State **Segment**

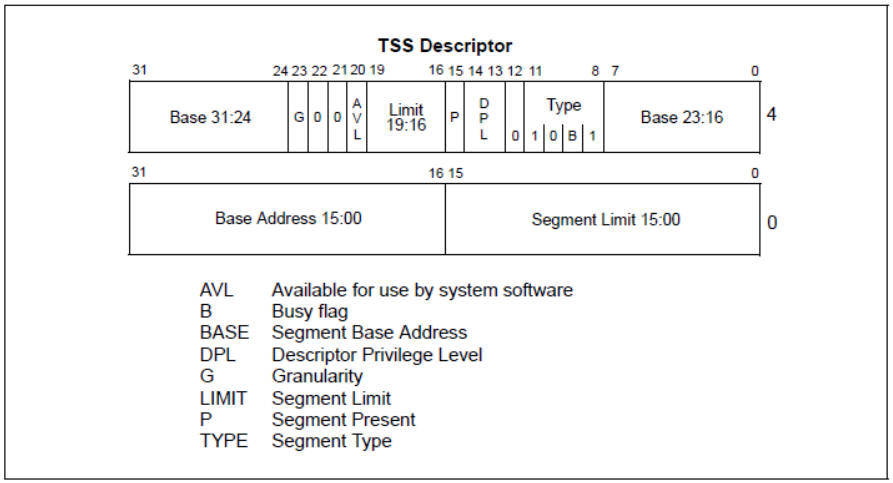



Figure 7-3. TSS Descriptor

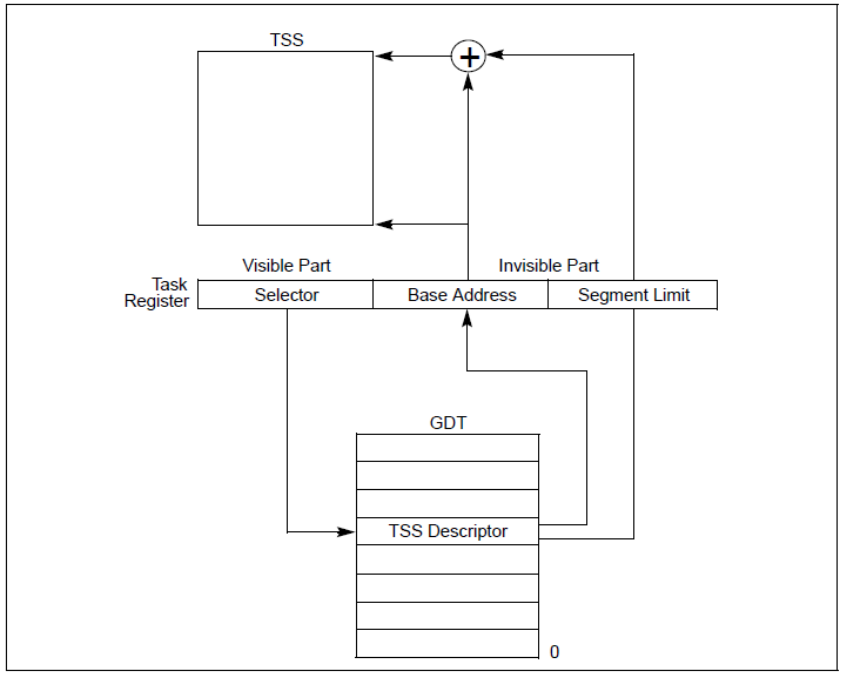



Figure 7-5. Task Register

# x86 privilege levels – TSS format

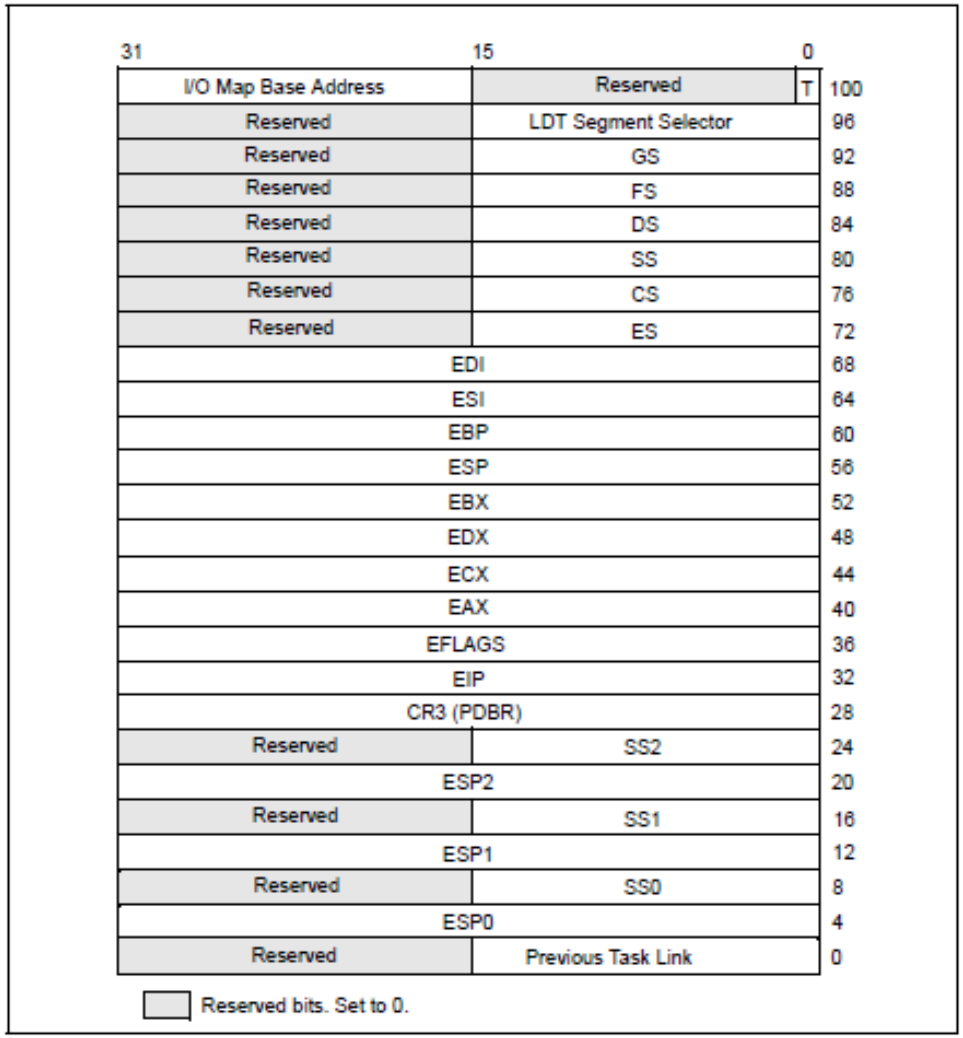
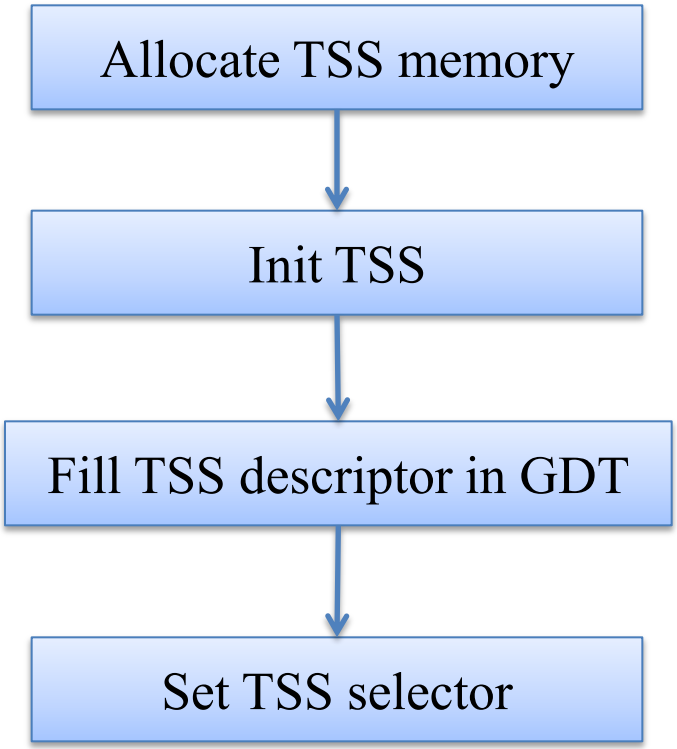



Figure 7-2. 32-Bit Task-State Segment (TSS)

# x86 privilege levels – establish TSS




pmm.c:27

pmm.c:80-81

pmm.c:84-85

pmm.c:91

# x86 privilege levels - References

- 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

- Know the format of page tables
- Know how segment/page tables are established
- Be able to operate on page table entries

# X86 HARDWARE MMU

# x86 hardware MMU – segmentation review

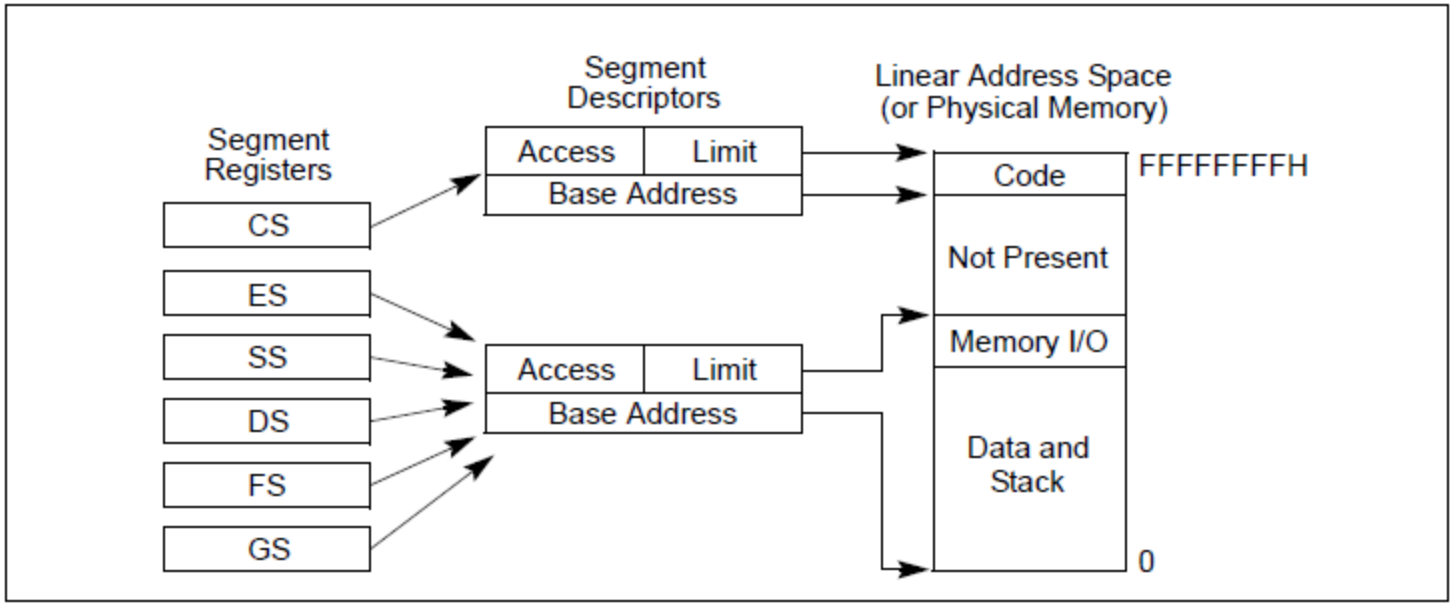
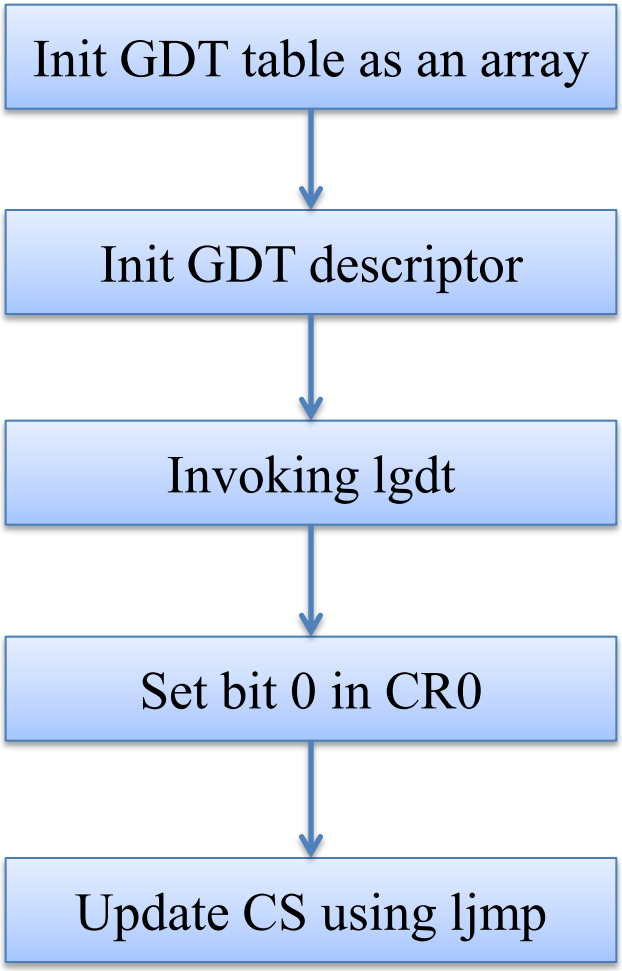



Figure 3-3. Protected Flat Model

# x86 hardware MMU – establish GDT tables (bootloader)




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

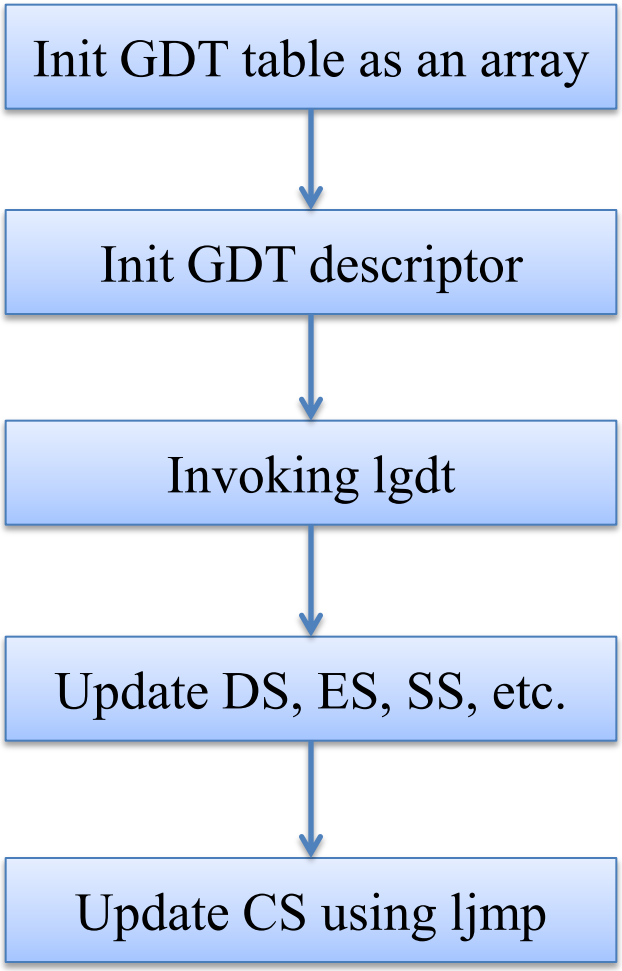
bootasm.S:105

bootasm.S:69

bootasm.S:70-72

bootasm.S:76

# x86 hardware MMU – establish GDT tables (kernel init)




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

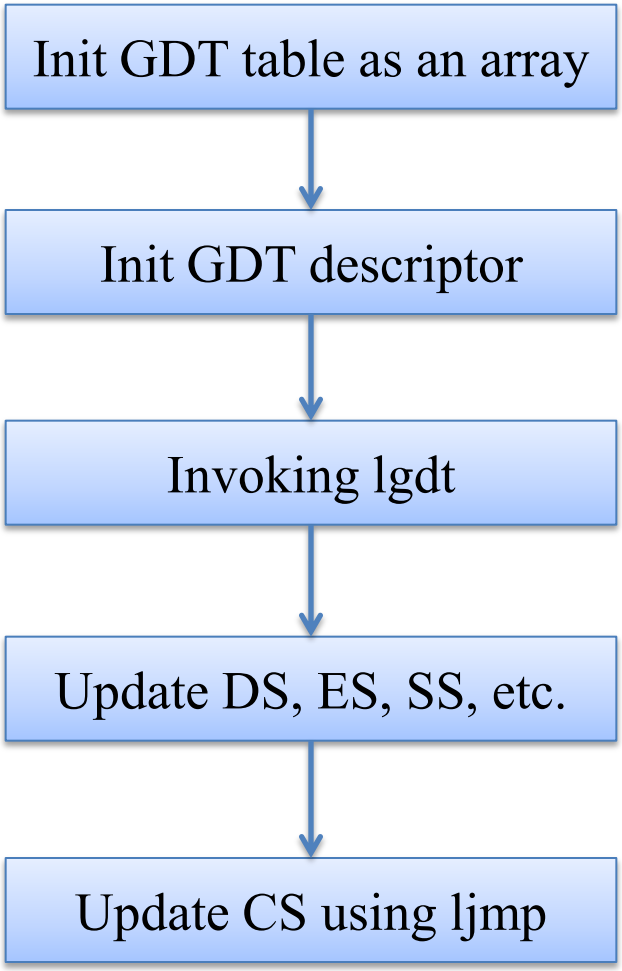
entry.S:46

entry.S:11

entry.S:12-15

entry.S:17

# x86 hardware MMU – establish GDT tables (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 hardware MMU – paging

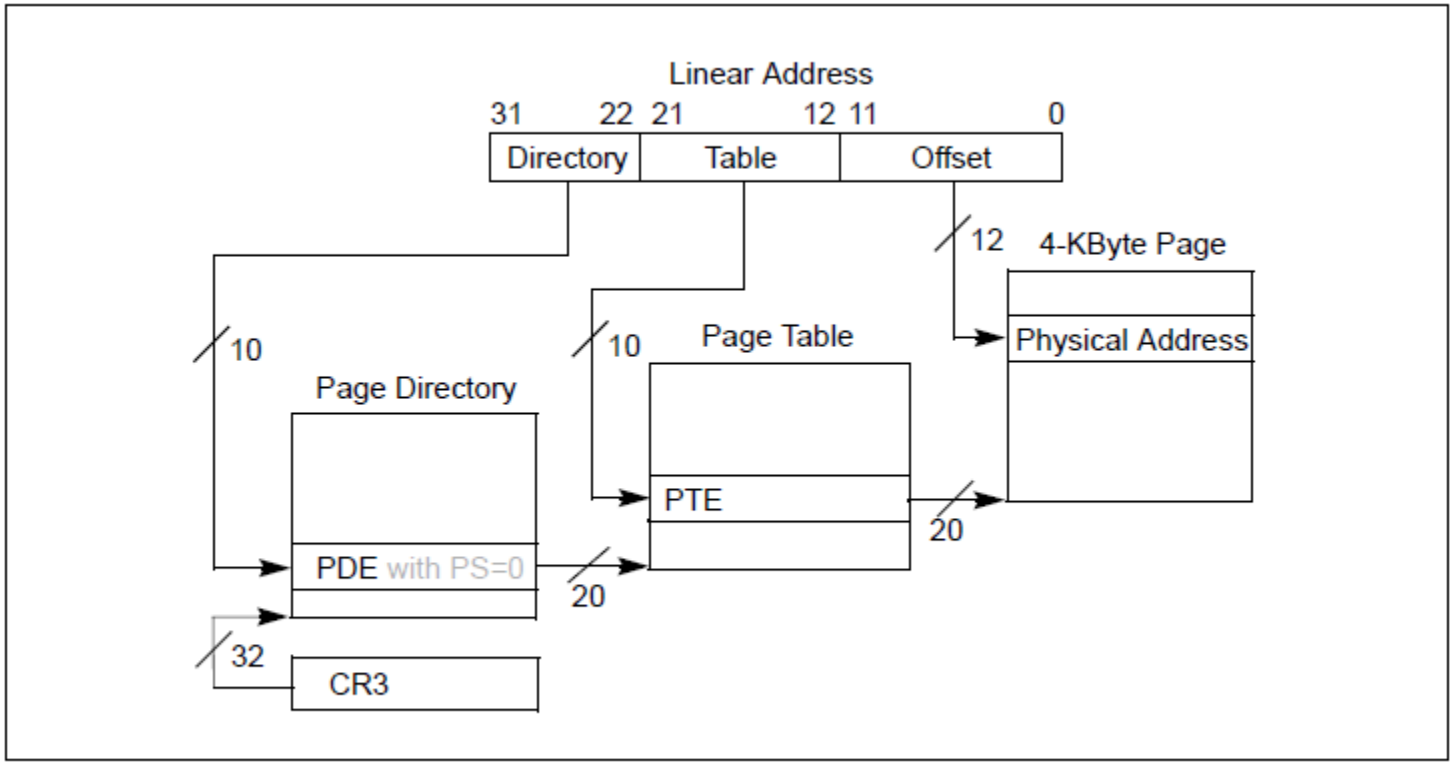
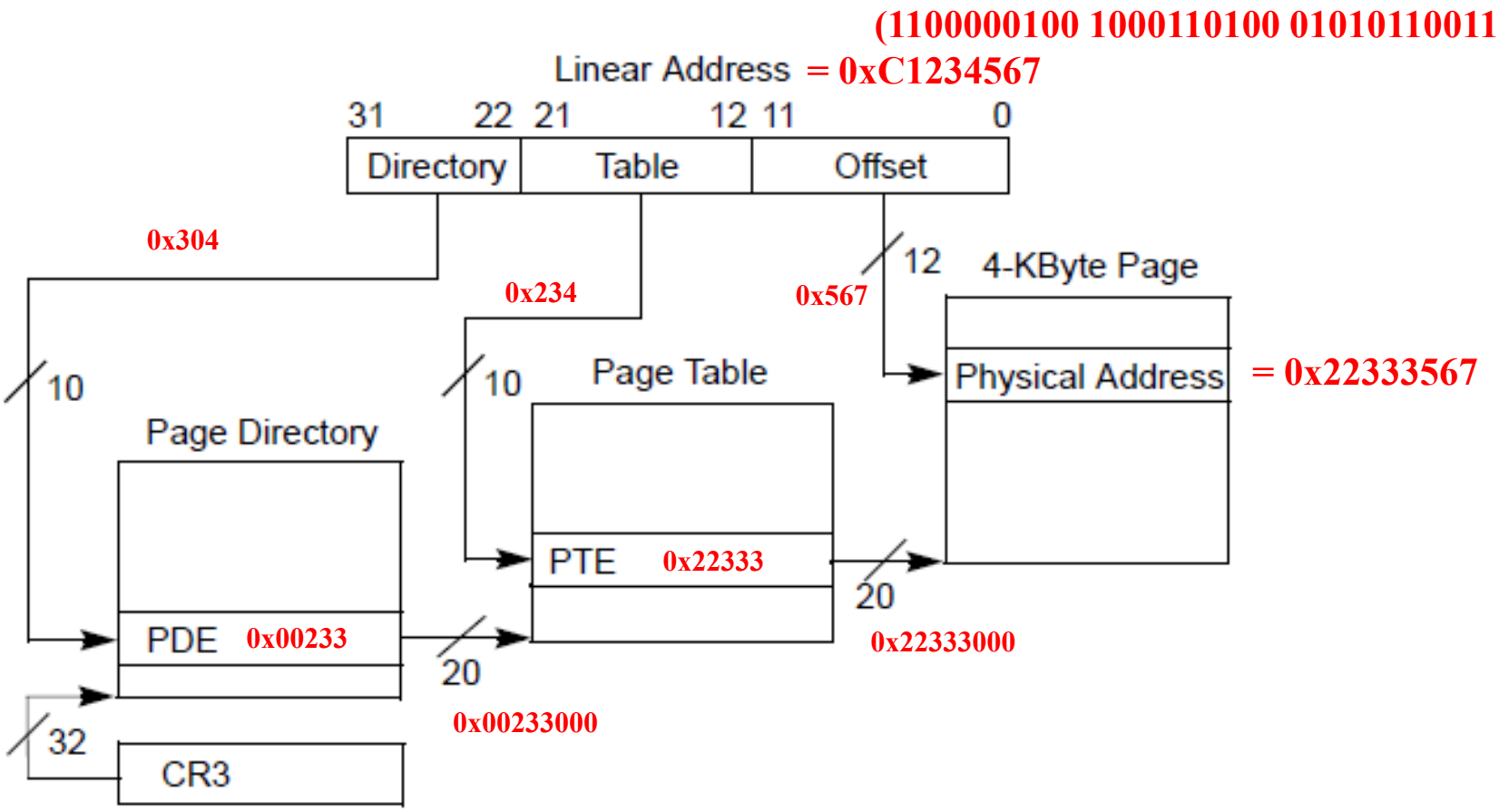



Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

# x86 hardware MMU – paging example




Addresses in page table entries are linear addresses!

# x86 hardware MMU – 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 hardware MMU – enable paging

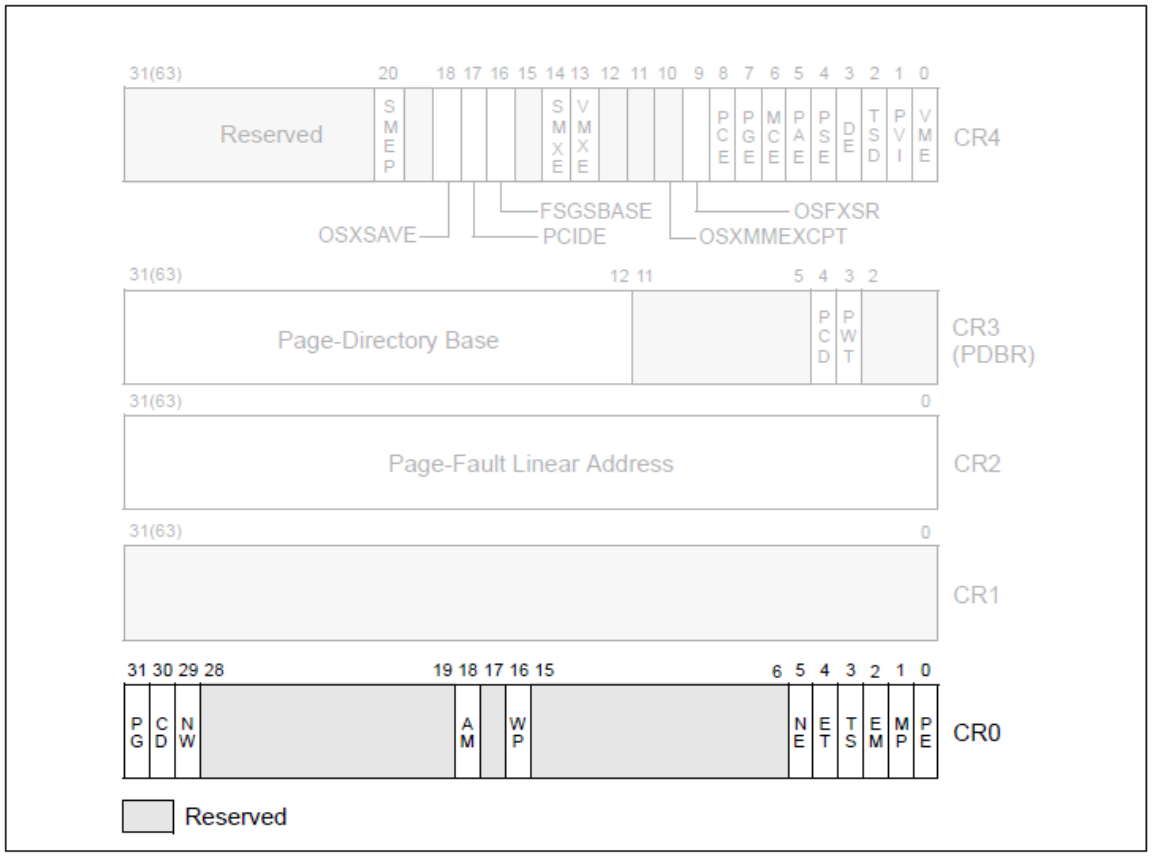
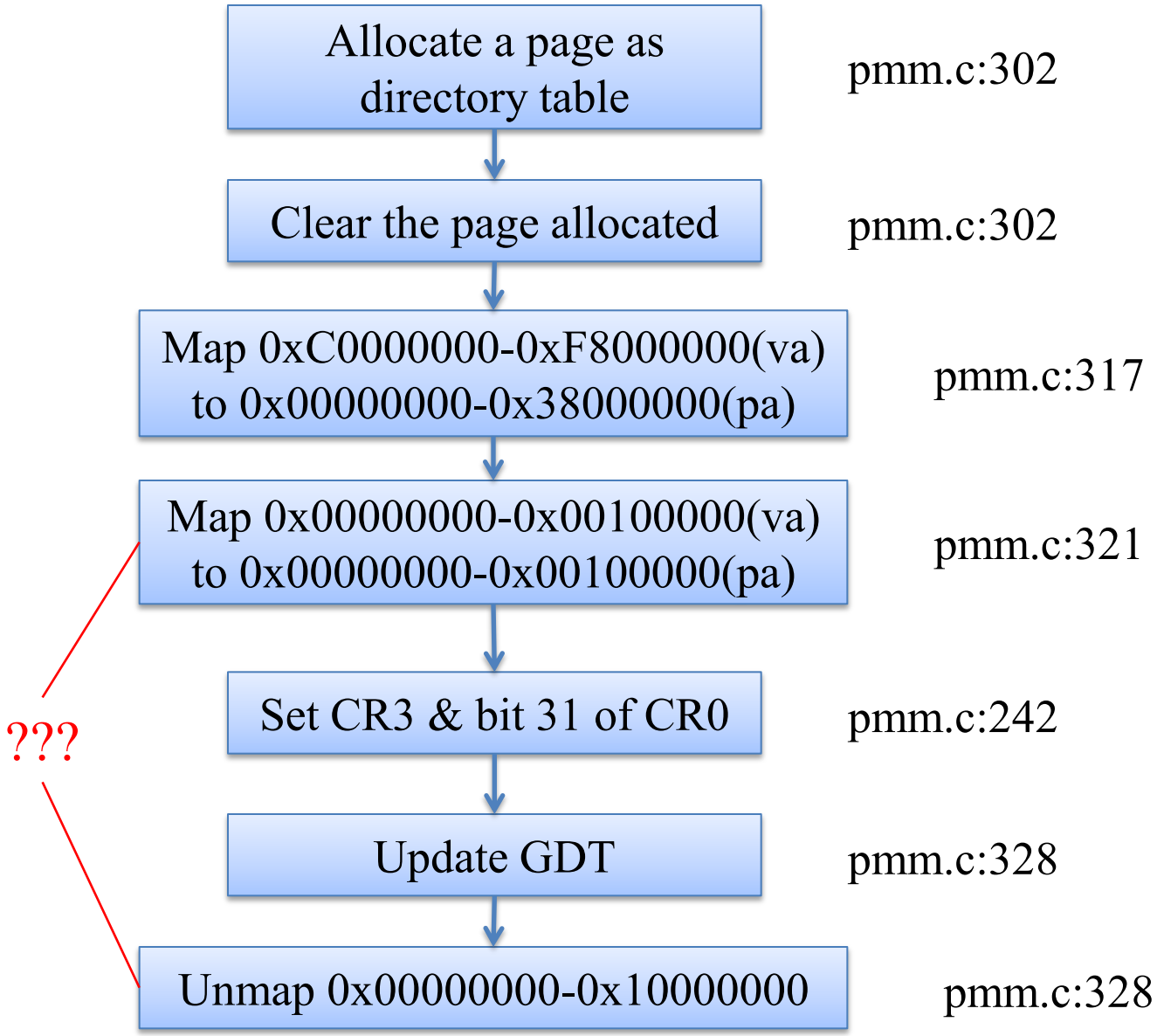


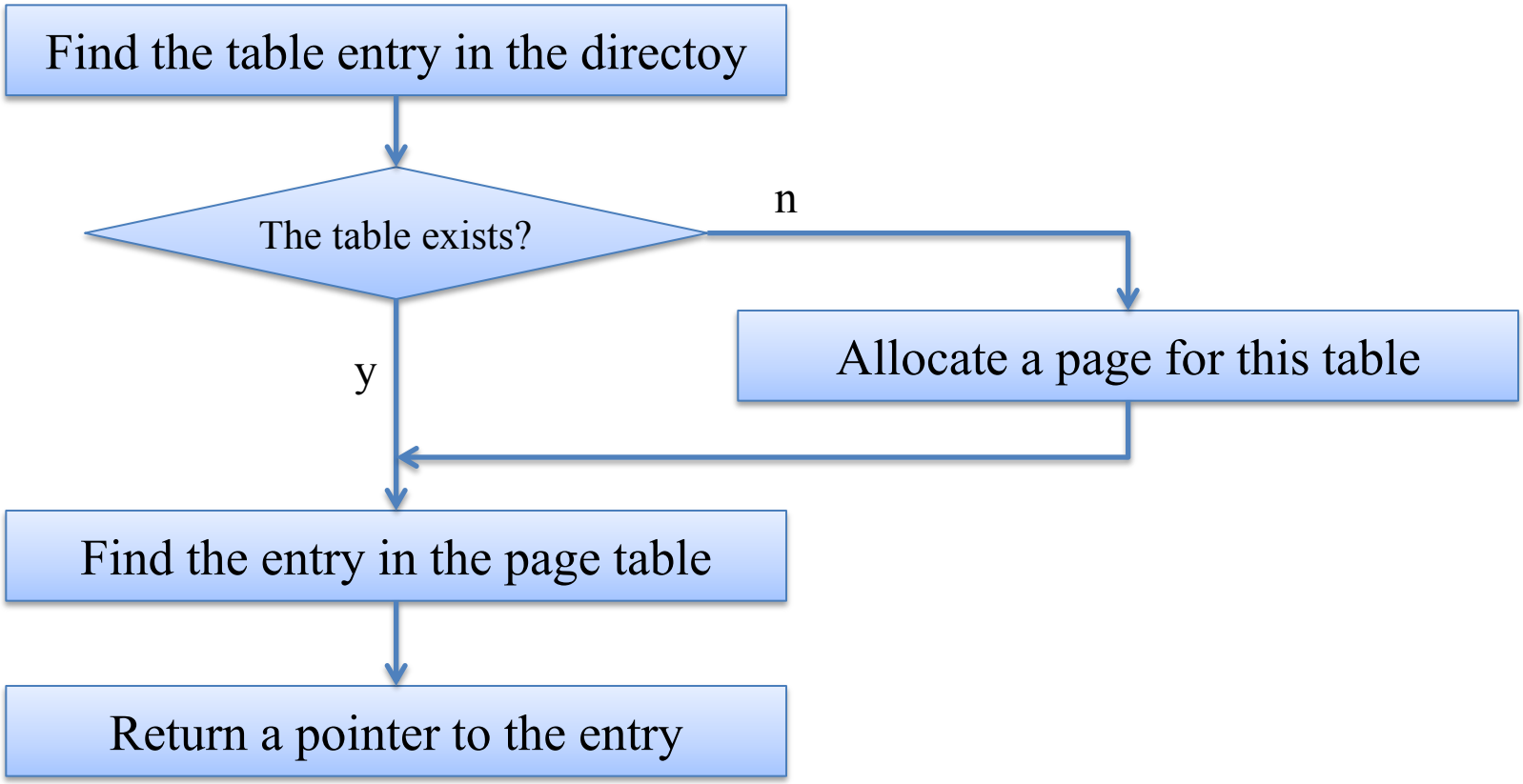

Figure 2-7. Control Registers

- To enable protection mode, OS should set bit 31 (PG) in CR0

# x86 hardware MMU – establish page tables



# x86 hardware MMU – map a page in the page table




YOUR WORK!

# x86 hardware MMU – merge segmentation & paging

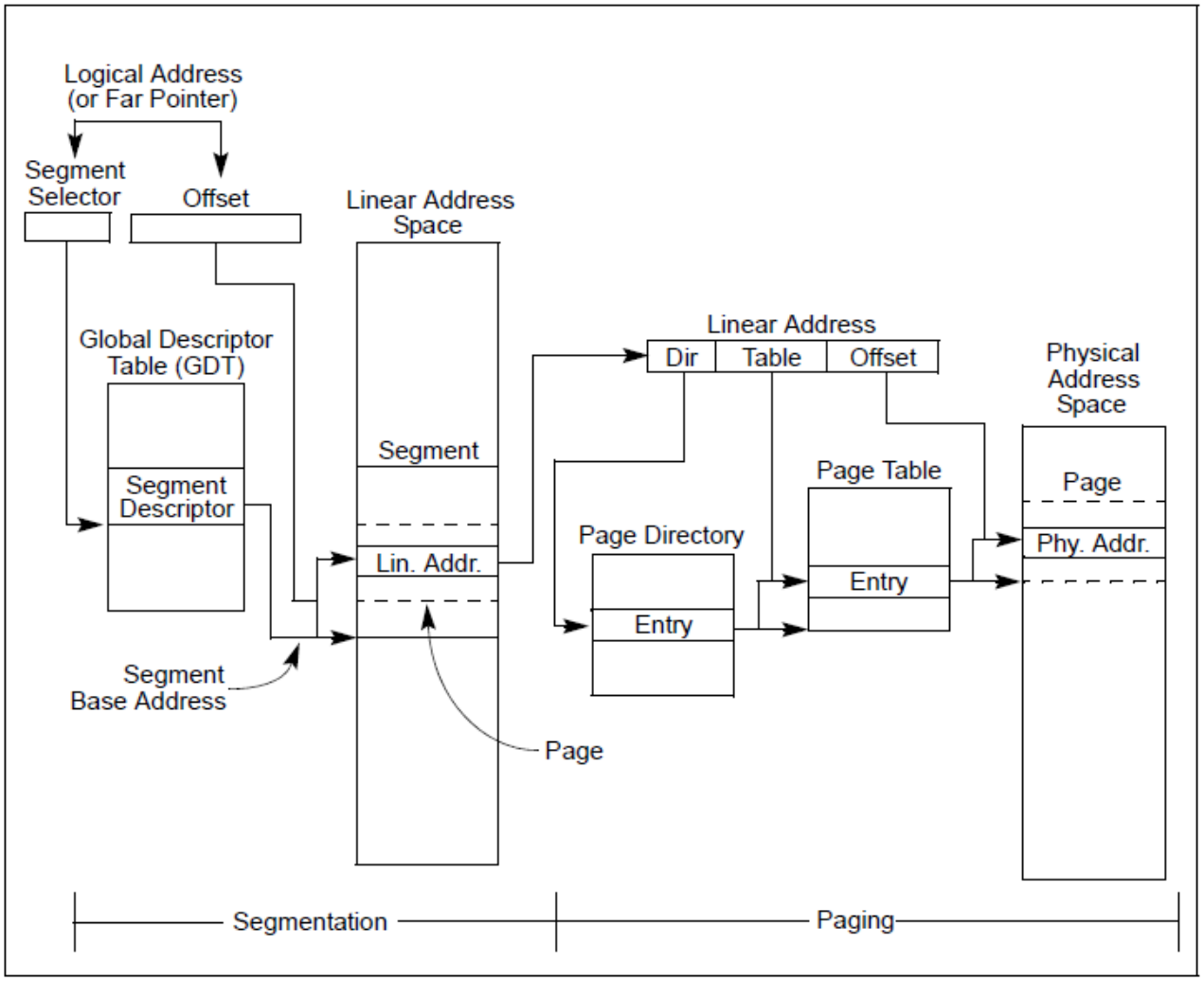
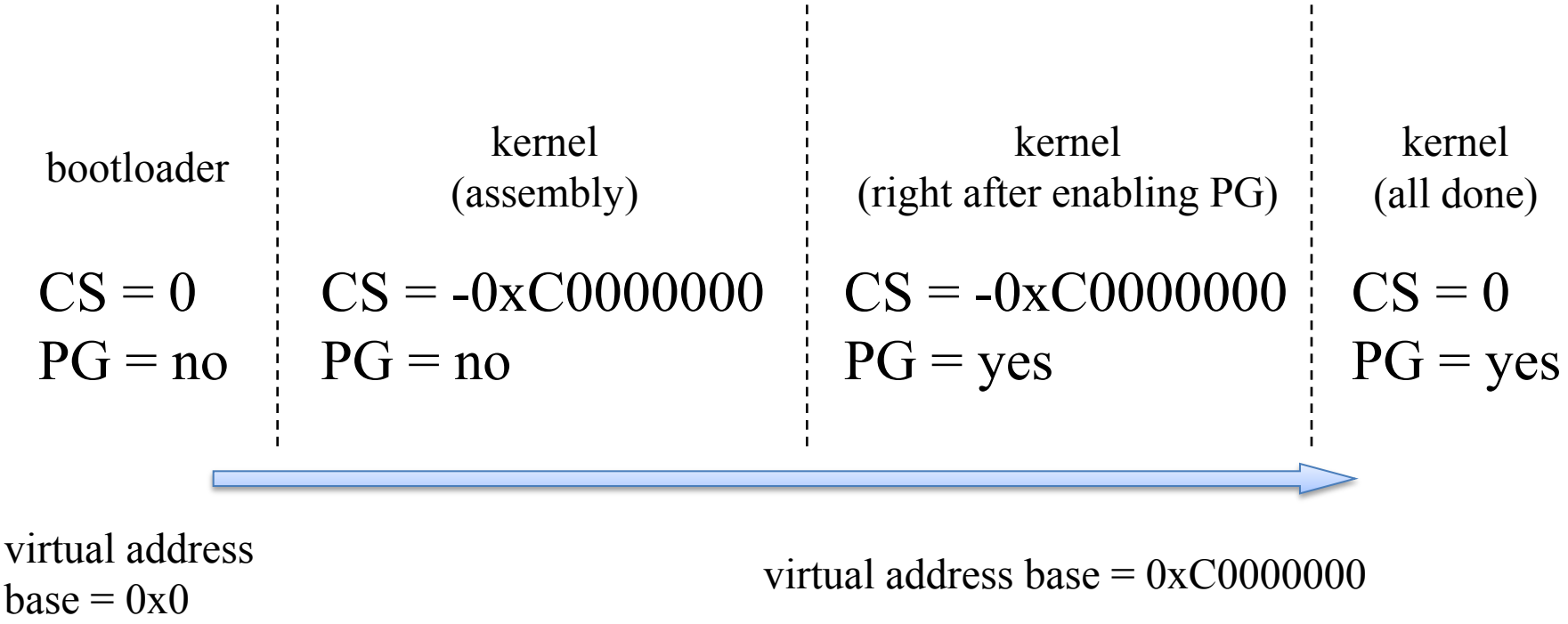
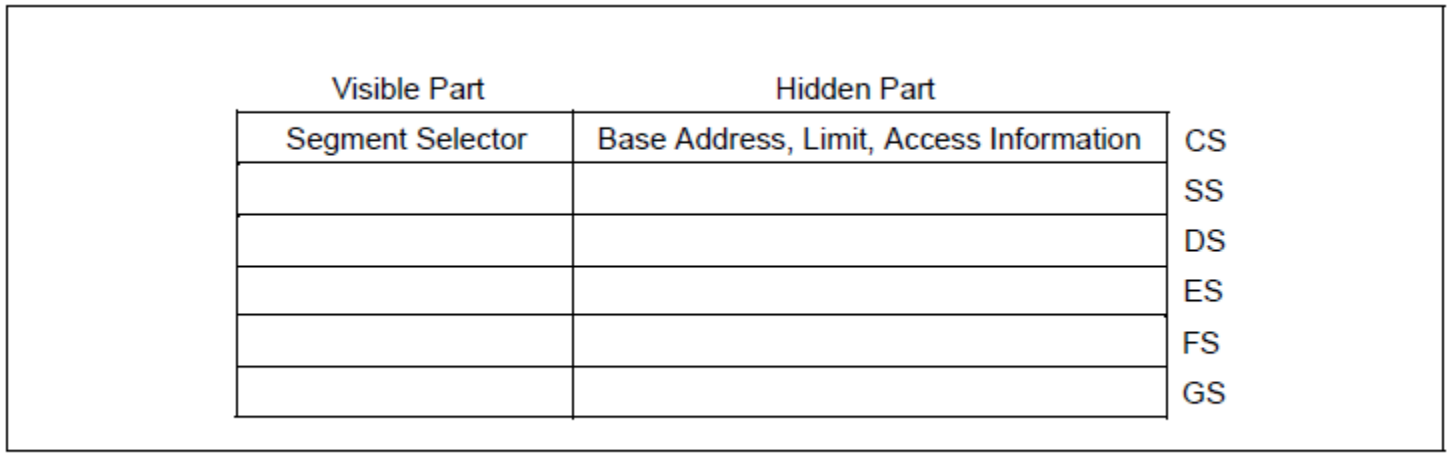



Figure 3-1. Segmentation and Paging

# x86 hardware MMU – memory management init in uCore



# x86 hardware MMU – hidden part in segment selectors



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

Figure 3-7. Segment Registers

- Base address are cached till next selector change.

# x86 hardware MMU – References

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

**That's all. Thanks!**