18-447: Computer Architecture
Lecture 18: Virtual Memory III

Yoongu Kim
Carnegie Mellon University
Spring 2013, 3/1

Upcoming Schedule

- Today: **Lab 3 Due**
- Today: **Lecture/Recitation**
- Monday (3/4): **Lecture** – Q&A Session
- Wednesday (3/6): **Midterm 1**
  - 12:30 – 2:20
  - Closed book
  - One letter-sized cheat sheet
    - Can be double-sided
    - Can be either typed or written

Readings

- Required
  - P&H, Chapter 5.4
  - Hamacher et al., Chapter 8.8

- Recommended
  - *Denning, P. J.* ***Virtual Memory.*** *ACM Computing Surveys. 1970*
  - *Jacob, B., & Mudge, T.* ***Virtual Memory in Contemporary Microprocessors.*** *IEEE Micro. 1998.*

- References
  - Intel Manuals for 8086/80286/80386/IA32/Intel64
  - MIPS Manual

Review of Last Lecture

- Two approaches to virtual memory
  1. **Segmentation**
     - Not as popular today
  2. **Paging**
     - What is usually meant today by “virtual memory”

- Virtual memory requires HW+SW support
  - HW component is called the **MMU**
    - Memory management unit
  - How to **translate**: virtual ↔ physical addresses?

Review of Last Lecture (cont'd)

1. Segmentation

- Divide the address space into segments
  - Physical Address = **`BASE`** + Virtual Address
- Case studies: Intel 8086, 80286, x86, x86-64
- **Advantages**
  - Modularity/Isolation/Protection
  - Translation is simple
- **Disadvantages**
  - Complicated management
  - Fragmentation
  - Only a few segments are addressable at the same time

Review of Last Lecture (cont'd)

2. Paging

- **Virtual address space:** Large, contiguous, imaginary
- **Page:** A fixed-sized chunk of the address space
- **Mapping:** Virtual pages → physical pages
- **Page table:** The data structure that stores the mappings
  - Problem #1: Too large
    - Solution: **Hierarchical** page tables
  - Problem #2: Large latency
    - Solution: **Translation Lookaside Buffer** (TLB)
- Case study: Intel 80386
- Today, we'll talk more about paging ...

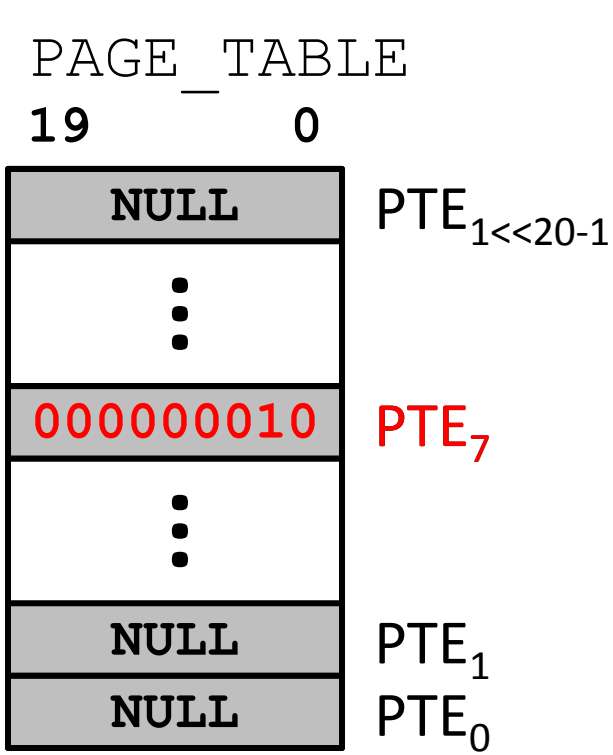
Today's Lecture

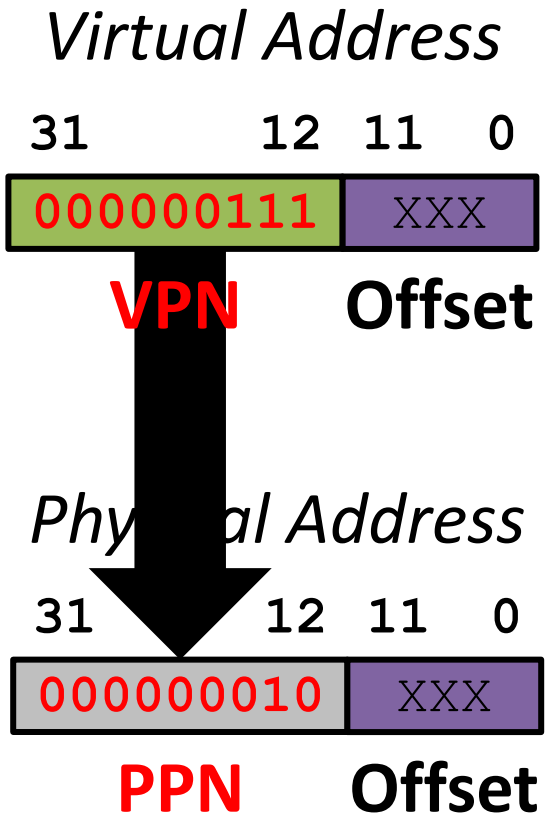
- More on Paging
  1. Translation
  2. Protection
  3. TLB Management
  4. Page Faults
  5. Page Size
  6. Software Side

1. TRANSLATION

Translation: “Flat” Page Table

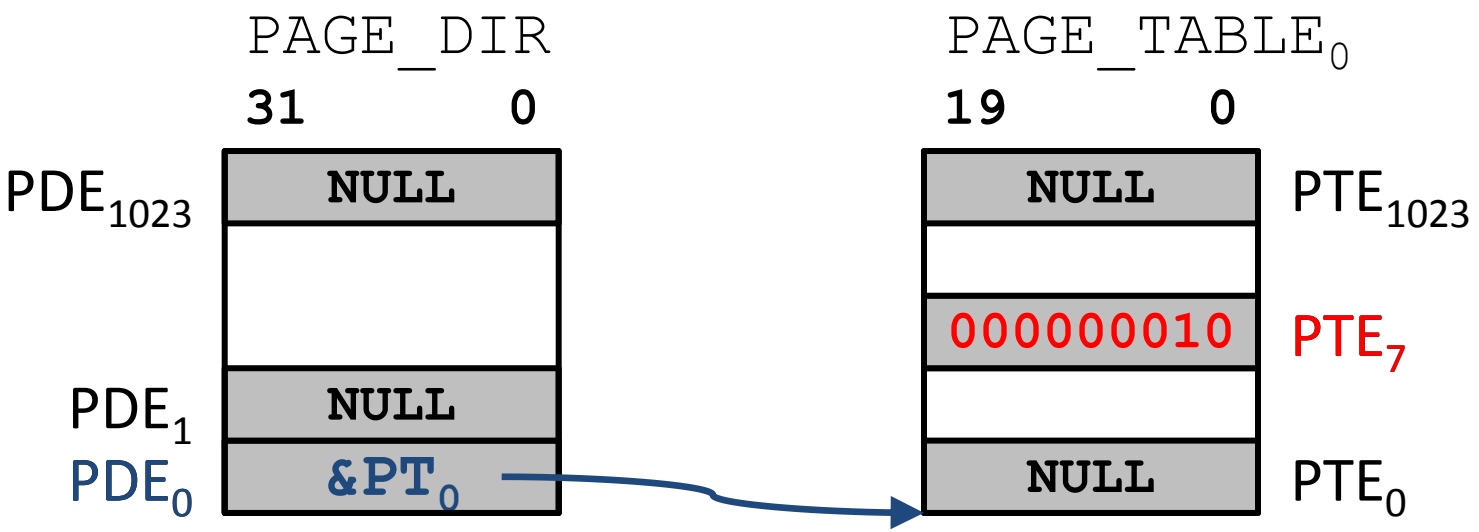
```
pte_t PAGE_TABLE[1<<20];
PAGE_TABLE[7]=2;
```





Translation: Two-Level Page Table

```
pte_t *PAGE_DIRECTORY[1<<10];
PAGE_DIRECTORY[0]=malloc((1<<10)*sizeof(pte_t));
PAGE_DIRECTORY[0][7]=2;
```



VPN[19:0]=0000000000_0000000111

Directory index *Table index*

Two-Level Page Table (x86)

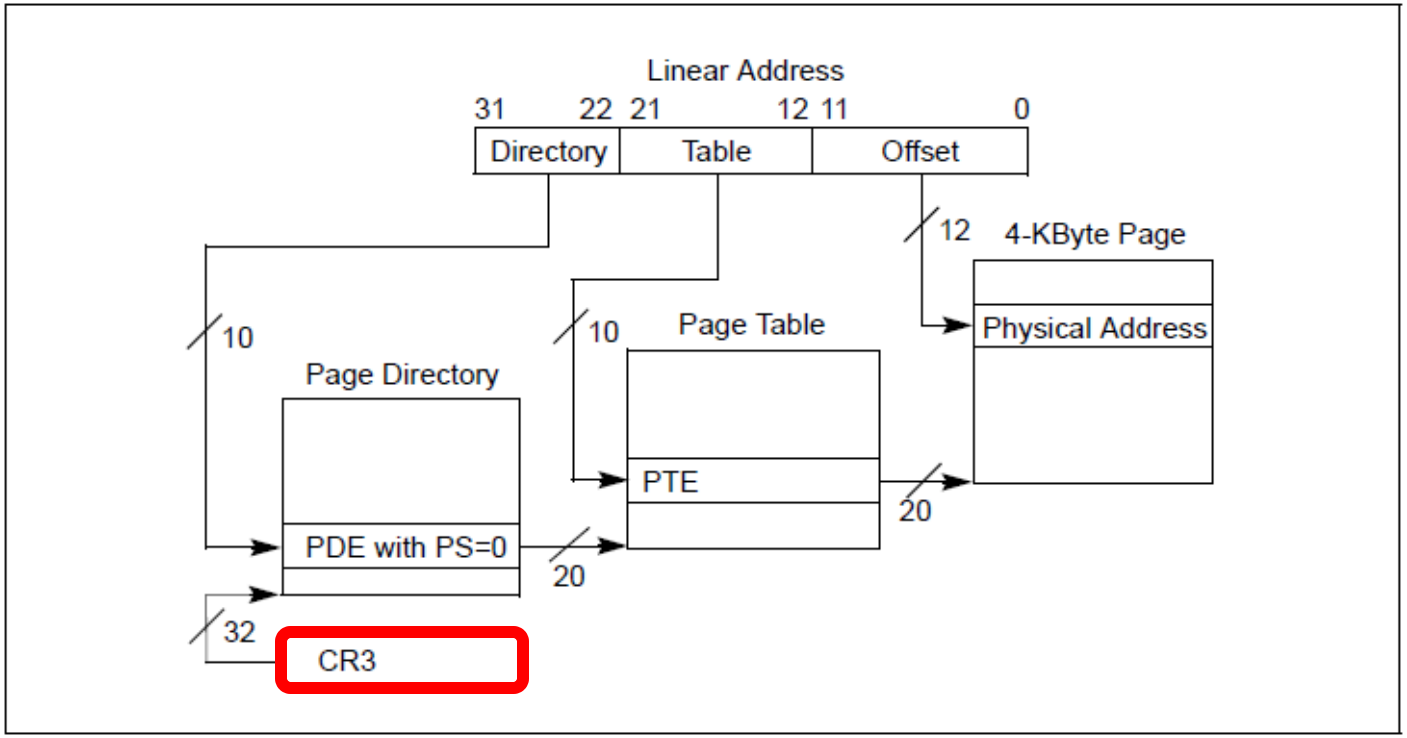
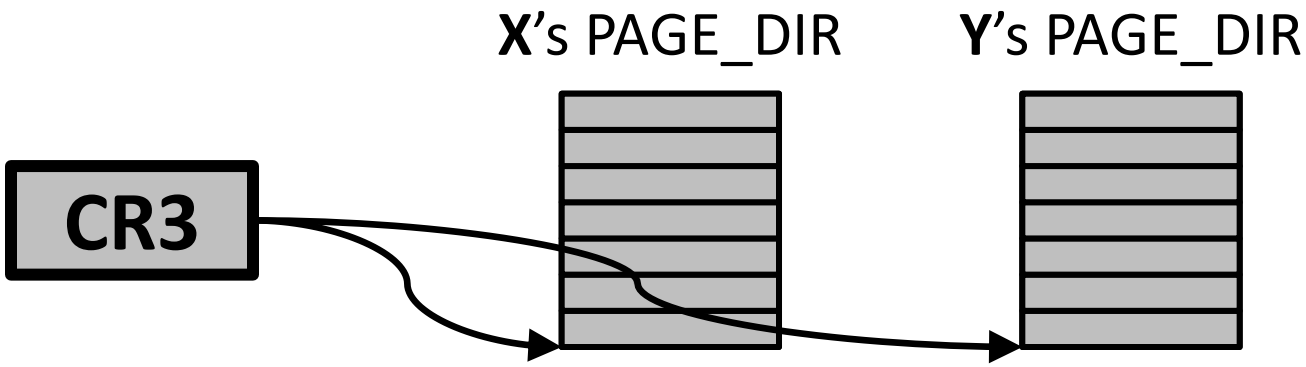


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

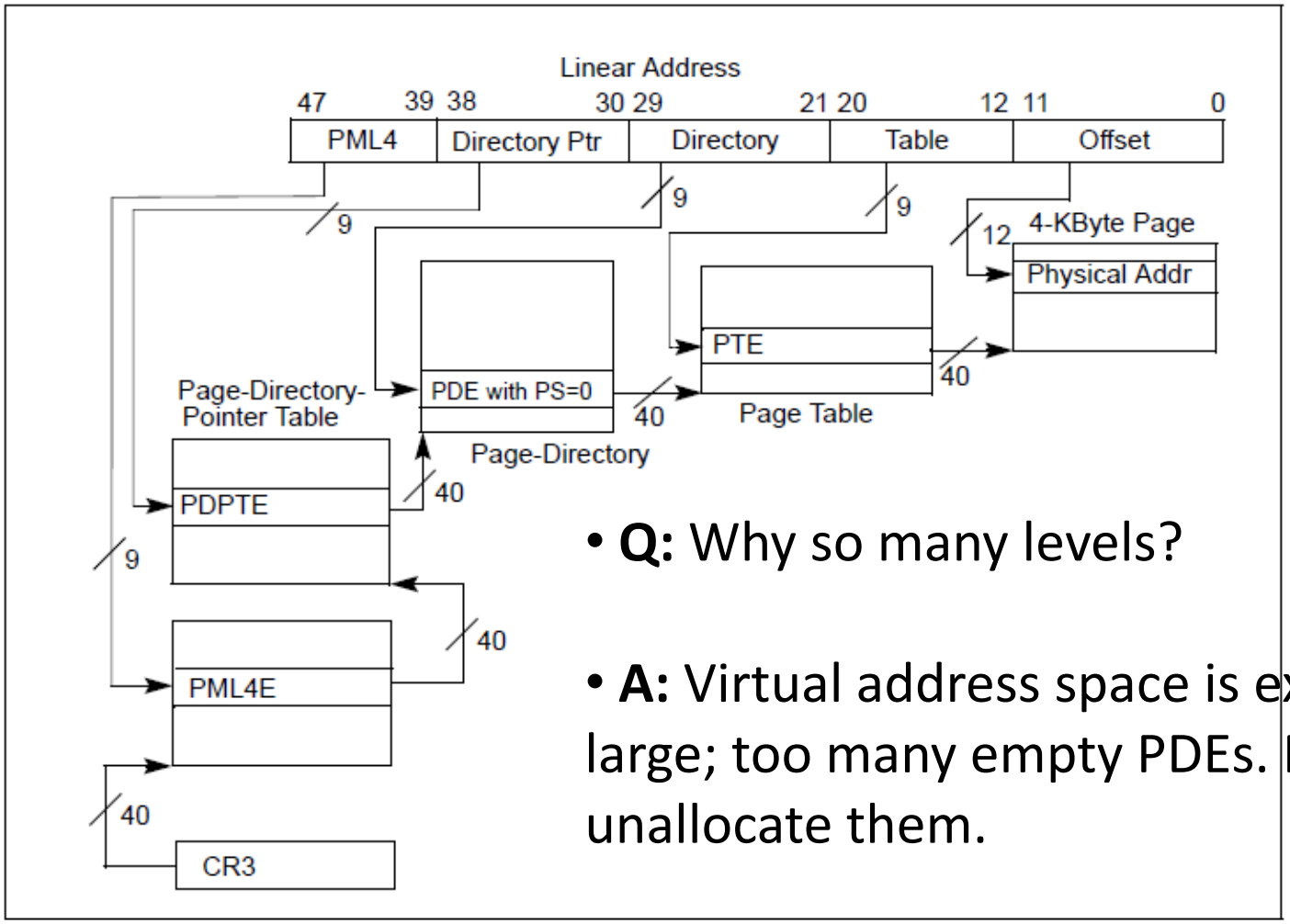
- **CR3**: Control Register 3 (or **Page Directory Base Register**)
  - Stores the physical address of the page directory
  - **Q:** Why not the virtual address?

Per-Process Virtual Address Space

- Each process has its own virtual address space
  - Process **X**: text editor
  - Process **Y**: video player
  - **X** writing to its virtual address 0 does not affect the data stored in **Y**'s virtual address 0 (or any other address)
    - This was the entire purpose of virtual memory
  - *Each process has its own page directory and page tables*
    - On a context switch, the CR3's value must be updated



Multi-Level Page Table (x86-64)



- **Q:** Why so many levels?

- **A:** Virtual address space is extremely large; too many empty PDEs. Need to unallocate them.

Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

Translation: Segmentation + Paging

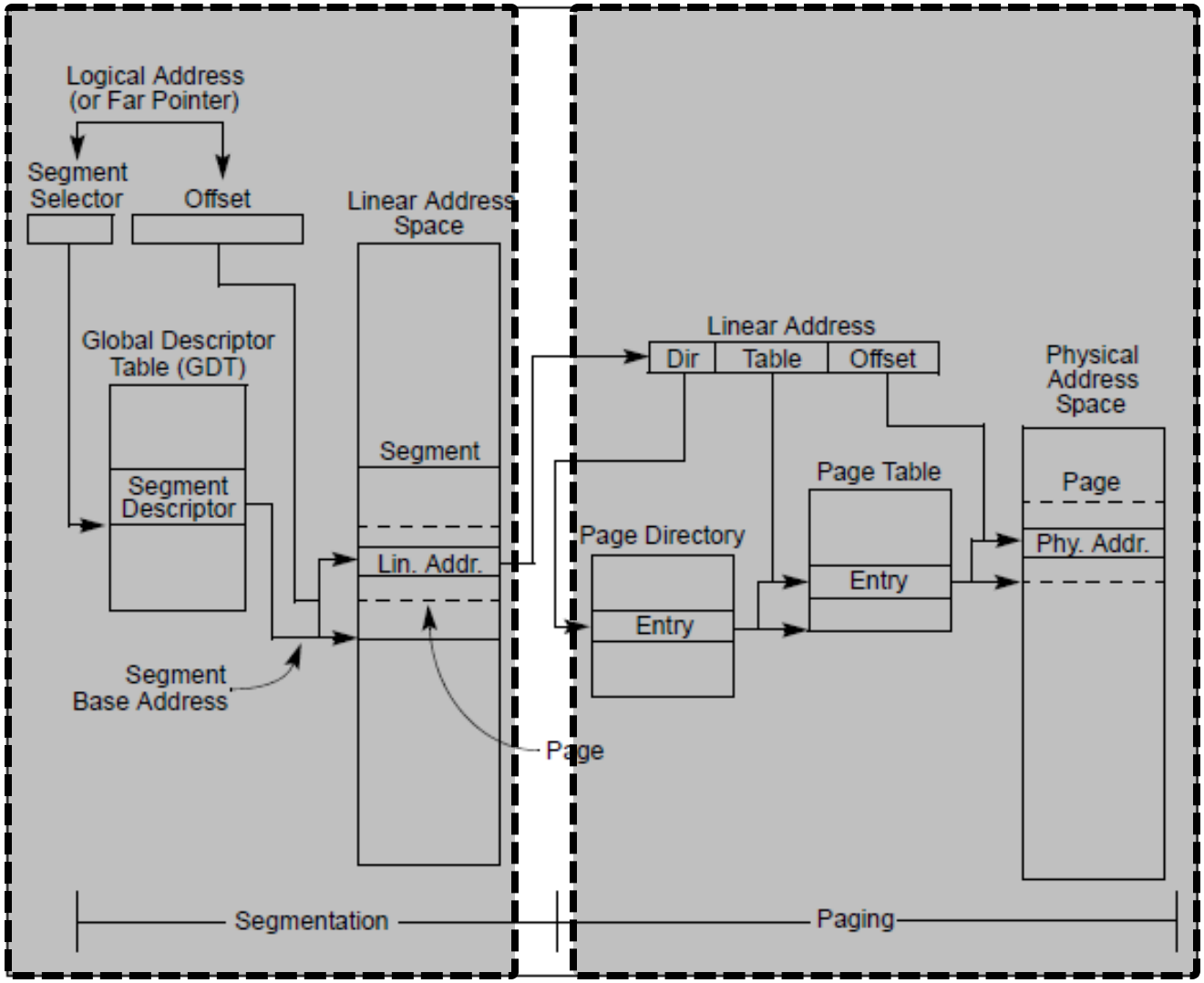


Figure 3-1. Segmentation and Paging

2. PROTECTION

x86: Privilege Level (Review)

- Four **privilege levels** in x86 (referred to as **rings**)
  - Ring 0: Highest privilege (operating system)
  - Ring 1: Not widely used **"Supervisor"**
  - Ring 2: Not widely used
  - Ring 3: Lowest privilege (user applications) **"User"**

- **Current Privilege Level** (CPL) determined by:
  - Address of the instruction that you are executing
  - Specifically, the **Descriptor Privilege Level** (DPL) of the code segment

x86: A Closer Look at the PDE/PTE

- **PDE:** Page Directory Entry (32 bits)
- **PTE:** Page Table Entry (32 bits)

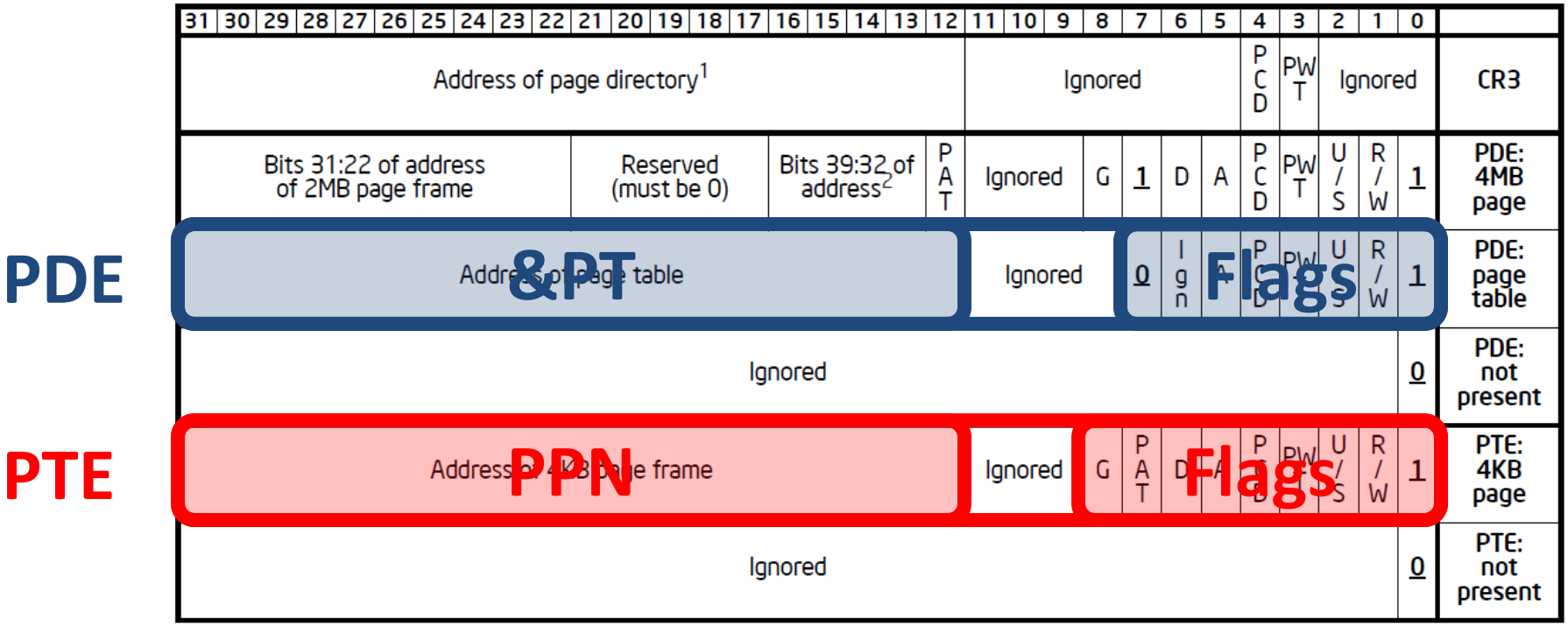


Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

Protection: PDE's Flags

- Protects all 1024 pages in a page table

Table 4-5. Format of a 32-Bit Page-Directory Entry that References a Page Table

Bit Position(s)	Contents
0 (P)	Present; must be 1 to reference a page table
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-MByte region controlled by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-MByte region controlled by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the page table referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether this entry has been used for linear-address translation (see Section 4.8)
6	Ignored
7 (PS)	If CR4.PSE = 1, must be 0 (otherwise, this entry maps a 4-MByte page; see Table 4-4); otherwise, ignored

Protection: PTE's Flags

- Protects one page at a time

Table 4-6. Format of a 32-Bit Page-Table Entry that Maps a 4-KByte Page

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	If the PAT is supported, indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2); otherwise, reserved (must be 0)[1]
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise

Protection: PDE + PTE = ???

Table 5-3. Combined Page-Directory and Page-Table Protection

Page-Directory Entry		Page-Table Entry		Combined Effect	
Privilege	Access Type	Privilege	Access Type	Privilege	Access Type
User	Read-Only	User	Read-Only	User	Read-Only
User	Read-Only	User	Read-Write	User	Read-Only
User	Read-Write	User	Read-Only	User	Read-Only
User	Read-Write	User	Read-Write	User	Read/Write
User	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
User	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	User	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	User	Read-Write	Supervisor	Read/Write
Supervisor	Read-Only	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Only	Supervisor	Read-Write	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Only	Supervisor	Read/Write*
Supervisor	Read-Write	Supervisor	Read-Write	Supervisor	Read/Write

NOTE:

* If CR0.WP = 1, access type is determined by the R/W flags of the page-directory and page-table entries. IF CR0.WP = 0, supervisor privilege permits read-write access.

Protection: Segmentation + Paging

- **Paging** provides protection
  - Flags in the PDE/PTE (x86)
    - Read/Write
    - User/Supervisor
    - Executable (x86-64)
- **Segmentation** also provides protection
  - Flags in the Segment Descriptor (x86)
    - Read/Write
    - Descriptor Privilege Level
    - Executable

5.12 COMBINING PAGE AND SEGMENT PROTECTION

When paging is enabled, the processor evaluates segment protection first, then evaluates page protection. If the processor detects a protection violation at either the segment level or the page level, the memory access is not carried out and an exception is generated. If an exception is generated by segmentation, no paging exception is generated.

3. TLB MANAGEMENT

TLB (Review)

- **Translation Lookaside Buffer** (TLB)
  - A hardware structure where PTEs are cached
    - **Q:** How about PDEs? Should they be cached?
  - Whenever a virtual address needs to be translated, the TLB is first searched: "**hit**" vs. "**miss**"

- Example: 80386
  - 32 entries in the TLB
  - TLB entry: **tag** + **data**
    - Tag: 20-bit VPN + 4-bit flag (valid, dirty, R/W, U/S)
    - Data: 20-bit PPN
    - **Q:** Why is the tag needed?

Context Switches

- Assume that Process **X** is running
  - Process **X**'s VPN 5 is mapped to PPN 100
  - The TLB caches this mapping
    - VPN 5 → PPN 100

- Now assume a context switch to Process **Y**
  - Process **Y**'s VPN 5 is mapped to PPN 200
  - When Process Y tries to access VPN 5, it searches the TLB
    - Process **Y** finds an entry whose tag is 5
    - Hurray! It's a TLB hit!
    - The PPN must be 100!
    - … Are you sure?

Context Switches (cont'd)

- Approach #1. Flush the TLB
  - Whenever there is a context switch, flush the TLB
    - All TLB entries are invalidated
  - Example: 80836
    - Updating the value of CR3 signals a context switch
    - This automatically triggers a TLB flush

- Approach #2. Associate TLB entries with processes
  - All TLB entries have an extra field in the tag ...
    - That identifies the process to which it belongs
  - Invalidate only the entries belonging to the old process
  - Example: Modern x86, MIPS

Handling TLB Misses

- The TLB is small; it cannot hold all PTEs
  - Some translations will inevitably miss in the TLB
  - Must access memory to find the appropriate PTE
    - Called **walking** the page directory/table
    - Large performance penalty

- Who handles TLB misses?
  1. Hardware-Managed TLB
  2. Software-Managed TLB

Handling TLB Misses (cont'd)

- Approach #1. **Hardware-Managed** (e.g., x86)
  - The hardware does the **page walk**
  - The hardware fetches the PTE and inserts it into the TLB
    - If the TLB is full, the entry **replaces** another entry
  - All of this is done transparently

- Approach #2. **Software-Managed** (e.g., MIPS)
  - The hardware raises an exception
  - The operating system does the **page walk**
  - The operating system fetches the PTE
  - The operating system inserts/evicts entries in the TLB

Handling TLB Misses (cont'd)

- Hardware-Managed TLB
  - Pro: No exceptions. Instruction just stalls
  - Pro: Independent instructions may continue
  - Pro: Small footprint (no extra instructions/data)
  - Con: Page directory/table organization is etched in stone

- Software-Managed TLB
  - Pro: The OS can design the page directory/table
  - Pro: More advanced TLB replacement policy
  - Con: Flushes pipeline
  - Con: Performance overhead

4. PAGE FAULTS

Introduction to Page Faults

- If a virtual page is not mapped to a physical page …
  - The virtual page does not have a valid PTE
  - x86: 0^{th} bit of PDE/PTE is set to 0

- What would happen if you accessed that page?
  - A hardware exception: **page fault**
  - The operating system needs to handle it
    - Page fault handler
  - *Side note*
    - *In x86, the term "page fault" has one additional meaning …*
    - *Violation of page protection (e.g., writing to read-only page)*
    - *For our purposes, we will not consider this to be a page fault*

Source of Page Faults

1. Program error

```
int *ptr=random();
int val=*ptr; // Error
```

   - The operating system cannot save you

2. The virtual page is mapped to **disk**, not memory
   - What is typically meant by “page fault”
   - The operating system can save you
     - Read the data from disk into a physical page in memory
     - Map the virtual page to the physical page
     - Create the appropriate PDE/PTE
     - Resume program that caused the page fault

Why Mapped to Disk?

- Why would a virtual page ever be mapped to disk?
  - Two possible reasons

1. Demand Paging

- When a large file in disk needs to be read, not all of it is loaded into memory at once
- Instead, page-sized chunks are loaded on-demand
- If most of the file is never actually read …
  - Saves time (remember, disk is extremely slow)
  - Saves memory space
- **Q:** When can demand paging be bad?

Why Mapped to Disk? (cont'd)

2. Swapping

- Assume that physical memory is exhausted
  - You are running many programs that require lots of memory
- What happens if you try to run another program?
  - Some physical pages are "swapped out" to disk
  - I.e., the data in some physical pages are migrated to disk
  - This frees up those physical pages
  - As a result, their PTEs become invalid
- When you access a physical page that has been swapped out, only then is it brought back into physical memory
  - This may cause another physical page to be swapped out
  - If this "ping-ponging" occurs frequently, it is called **thrashing**
  - Extreme performance degradation

5. PAGE SIZE

Trade-Offs in Page Size

- **Large page size (e.g., 1GB)**
  - Pro: Fewer PTEs required ➔ Saves memory space
  - Pro: Fewer TLB misses ➔ Improves performance
  - Con: Cannot have fine-grained permissions
  - Con: Large transfers to/from disk
    - Even when only 1KB is needed, 1GB must be transferred
    - Waste of bandwidth/energy
    - Reduces performance
  - Con: **Internal fragmentation**
    - Even when only 1KB is needed, 1GB must be allocated
    - Waste of space
    - **Q**: What is **external fragmentation**?

6. SOFTWARE SIDE

Hardware and Software

- Virtual memory requires both HW+SW support

- The hardware component is called the **MMU**
  - Most of what's been explained today is done by the MMU

- It is the job of the software to leverage the MMU
  - Populate page directories and page tables
  - Modify the Page Directory Base Register on context switch
  - Set correct permissions
  - Handle page faults
  - Etc.

More on Software

- Other software issues (that we won't go into)
  - Keeping track of which physical pages are free
  - Allocating free physical pages to virtual pages
  - Page replacement policy
    - When no physical pages are free, which should be swapped out?
  - Sharing pages between processes
  - Copy-on-write optimization
  - Page-flip optimization

Today's Lecture

- More on Paging
  1. Translation
  2. Protection
  3. TLB Management
  4. Page Size
  5. Page Faults
  6. Software Side