Project 5: Virtual Memory

Project Info

- Design Reviews: Tuesday, 8am-12pm, 11/25
  - You must do these before you leave for break
- OH: Thursday, 11/20, 4:30-6:30pm and 7:30-9:30pm. Will probably host second round of OH on Sunday after break
- Due date: Thursday, 12/4, at 11:59pm

General Notes

- You must implement everything before you can run the tests
- Familiarize yourself with the 2-level page table description of i386
  - Read section 3.7.1 and 4.2 of the Intel manual, linked off project website
- Inspect new PCB structure in kernel.h

High-level bits

- Set up memory for the kernel
- Set up virtual memory for each process, done in the kernel when you create a new process
  - Each process now runs in virtual memory. Mapping virtual memory-> physical memory is now the responsibility of the kernel. Hardware then uses this mapping when instructions are actually executed.
- Implement the page_fault_handler() in the kernel
  - If a virtual page is not in memory, the kernel **pages it in** from disk, and it becomes mapped to a physical page. Physical pages are static; virtual pages are moved between physical memory and disk.

Mapping virtual → physical

(Straight from manual) To select the various table entries, the linear address is divided into three sections:

- (**Level 1**) Page-directory entry—Bits 22 through 31 provide an offset to an entry in the page directory. The selected entry provides the base physical address of a page table.
- (**Level 2**) Page-table entry—Bits 12 through 21 of the linear address provide an offset to an entry in the selected page table. This entry provides the base physical address of a page in physical memory.
- Page offset—Bits 0 through 11 provides an offset to a physical address in the page.

A Picture

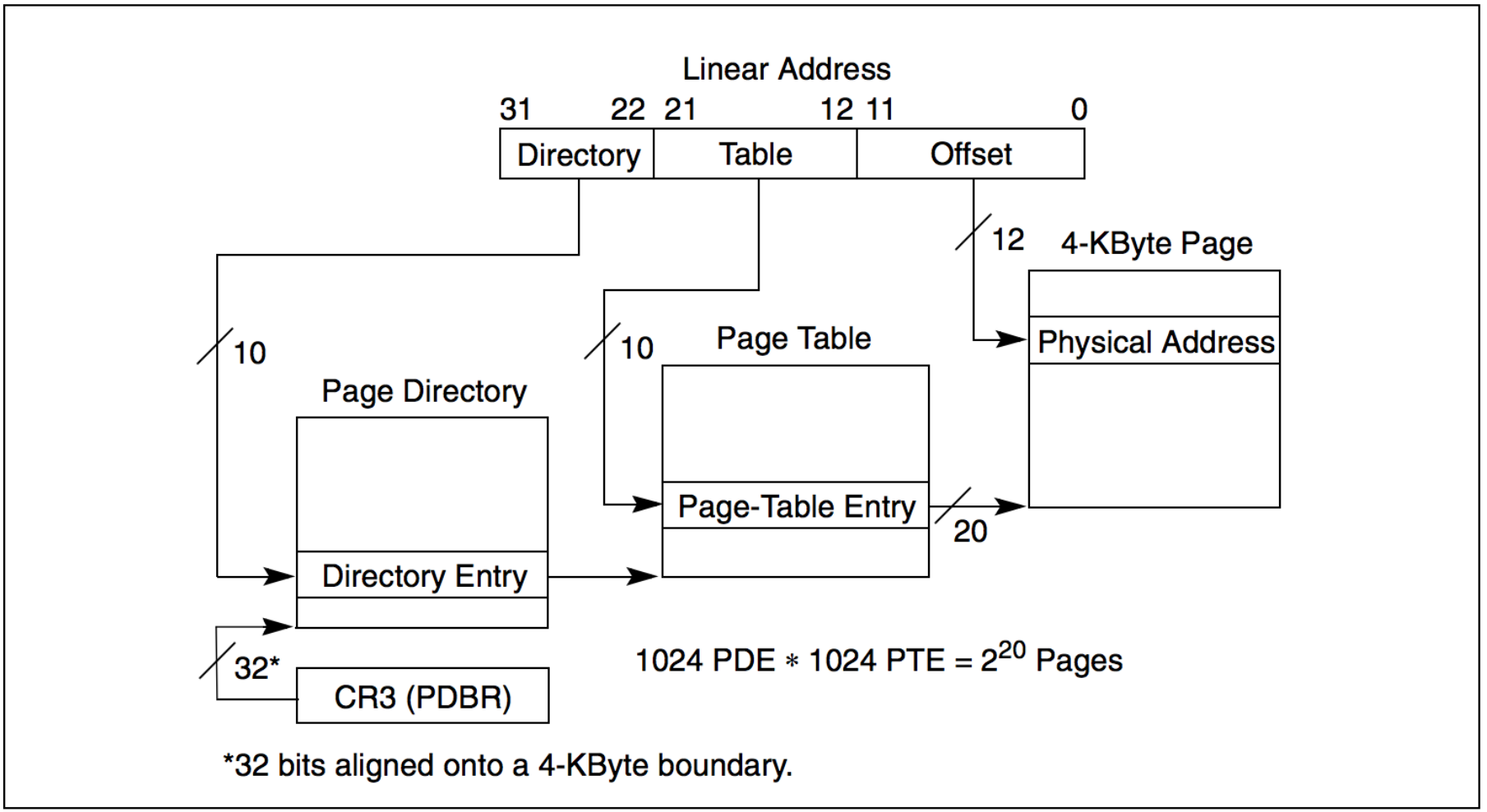
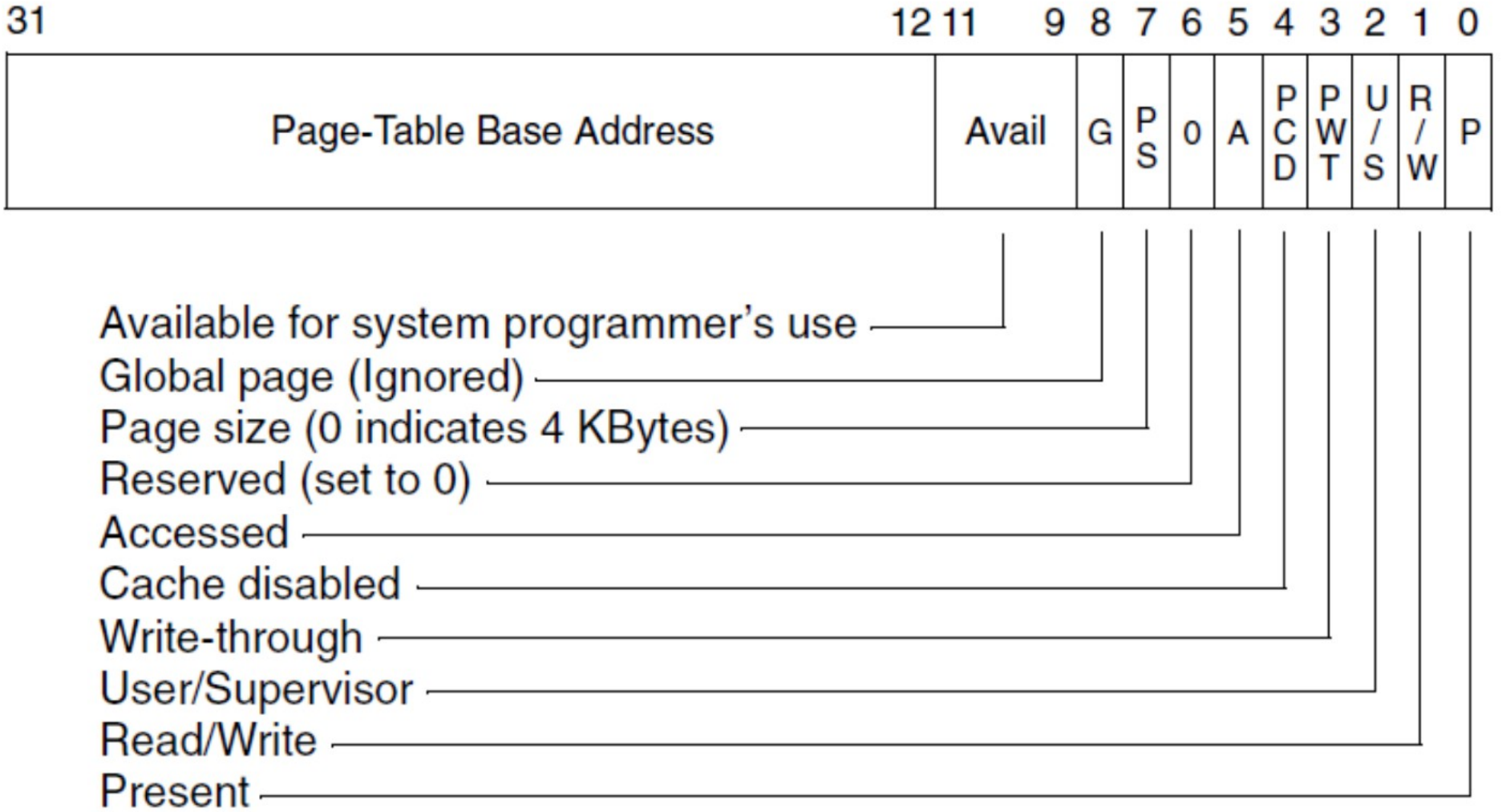
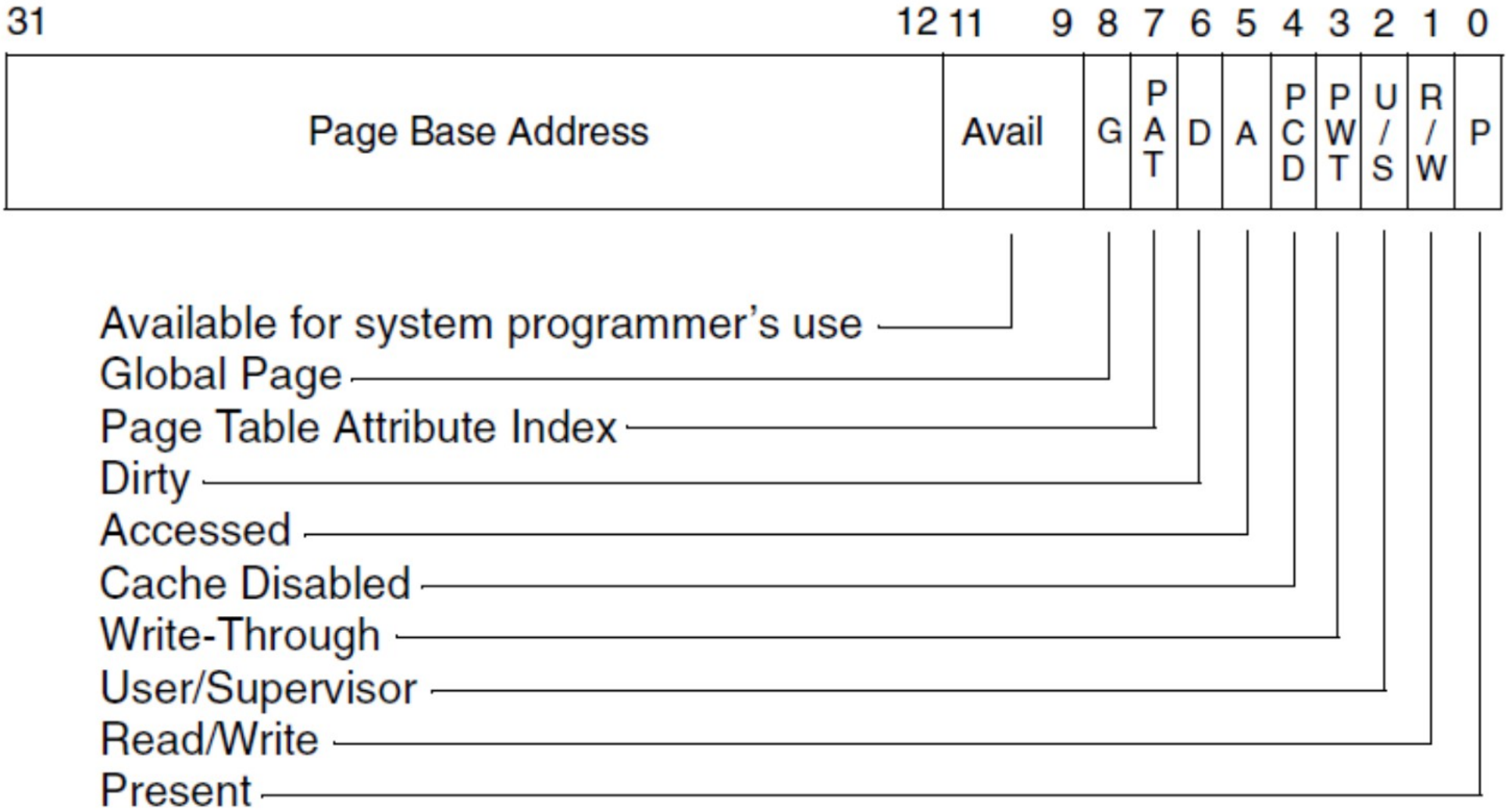


Figure 3-12. Linear Address Translation (4-KByte Pages)

Directory Entry



Page Entry


31
12
11
9
8
7
6
5
4
3
2
1
0
Page Base Address
Avail
G
PAT
D
A
PCD
PWT
U/S
R/W
P
Available for system programmer's use
Global Page
Page Table Attribute Index
Dirty
Accessed
Cache Disabled
Write-Through
User/Supervisor
Read/Write
Present

Page Faults

- Page fault happens because virtual page is not resident on a physical page
- How does hardware know a page fault happened?
- Need to keep track of metadata of physical pages
  - Free or not?
  - Metadata so you can do a replacement policy (FIFO sufficient for this assignment)
  - Pinned? When would you want to pin a phys page?

Paging from disk

- To resolve a page fault, might have to evict contents of a physical page to disk
  - Then need 2 disk ops: 1) eviction of contents of physical page, 2) bring in contents of virtual page, which are sitting on disk, and copy contents into the physical page
- Actual code uses a USB disk image for swap storage (usb/scsi.h)
- Assume that processes do not change size (no dynamic memory allocation)
- When to flush TLB?
- Update page tables!

Initializing Kernel Memory

- Allocate N_KERNEL_PTS (page tables)
- For each page table, allocate pages until you reach MAX_PHYSICAL_MEMORY
- For the kernel, **physical address == virtual address**
- Set correct flags
  - Give user permission to use the memory pages associated with the screen

Setting up Process Memory

- Processes keep track of 4 types of pages:
  - Page directory
  - Page tables
  - Stack page table
  - Stack pages
- PROCESS_START (virtual address of code + data)
  - Use one page table and allocate all pages
  - Process gets pcb->swap_size memory
- PROCESS_STACK (vaddr of stack top)
  - Allocate N_PROCESS_STACK_PAGES

More Tips

- One page table is enough for a process' code and data memory space.
- Some functions (esp the page fault handler) can be interrupted!
  - Use sync primitive
- Some pages don't need to be swapped out
  - Kernel pages, process page directory, page tables, stack page tables, and stack pages
- Project website can be confusing about the term “pages”. Use common sense to determine if it means physical or virtual page.