Operating Systems
CMPSC 473

Page Tables Recap / Project 2
Lecture 15: October 10, 2023

Instructor: Ruslan Nikolaev

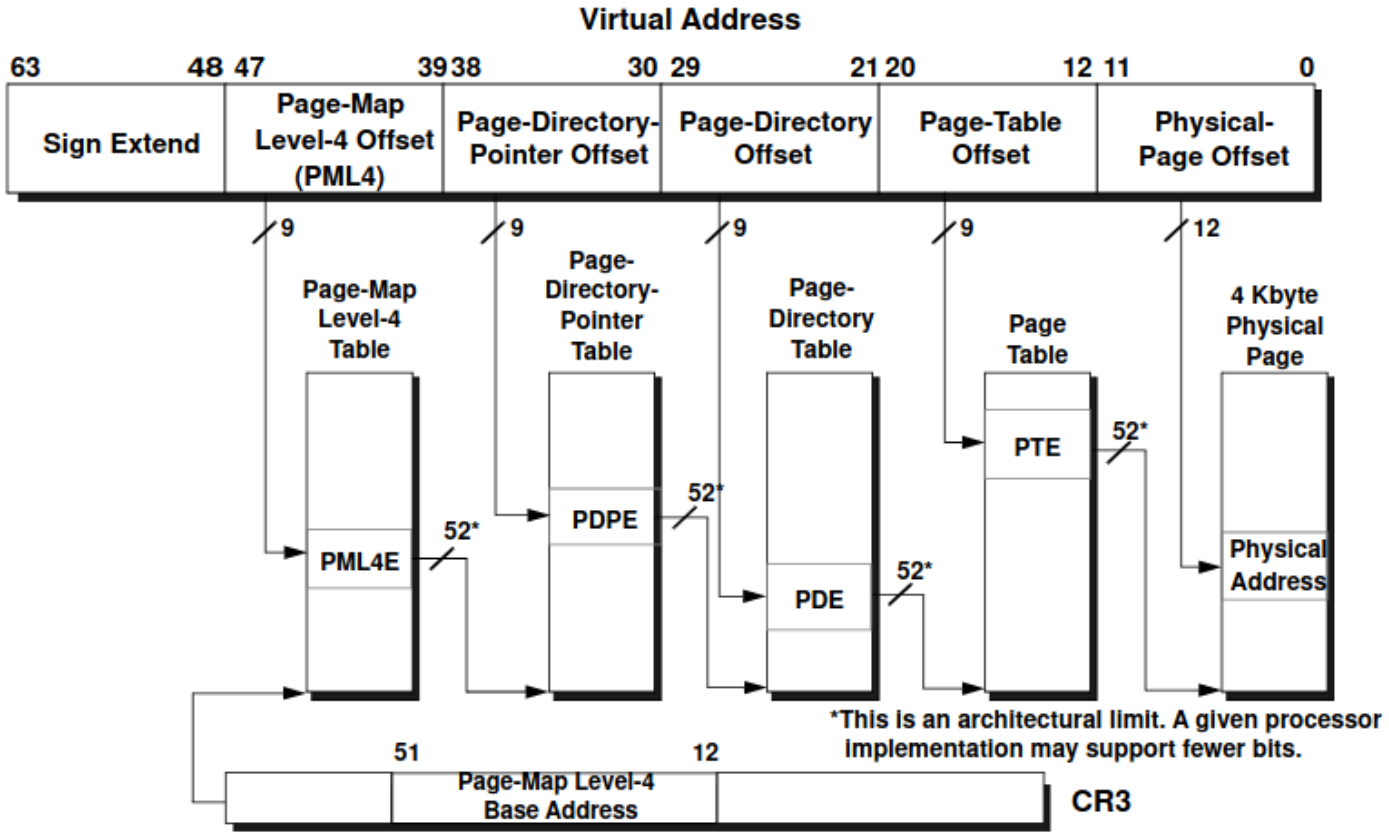
Question

- We have learnt that, in addition to the VPN->PPN translation, there is variety of other information stored in a page table: valid bit, permissions, kernel/user accessibility. Another similar piece of information is the so-called "dirty" bit which is set if a store occurs within a page. What benefit do you think this brings?

Question

- Consider information stored within page tables other than address translation (permissions, reference, dirty, valid). Select all that apply for a k-level page table (k > 1):
  - These bits need to be stored at each level
  - These bits need to be stored only at level 1
  - These bits need to be stored only at level k

Example: x86-64



* The picture is taken from https://www.amd.com/system/files/TechDocs/24593.pdf

Example: x86-64

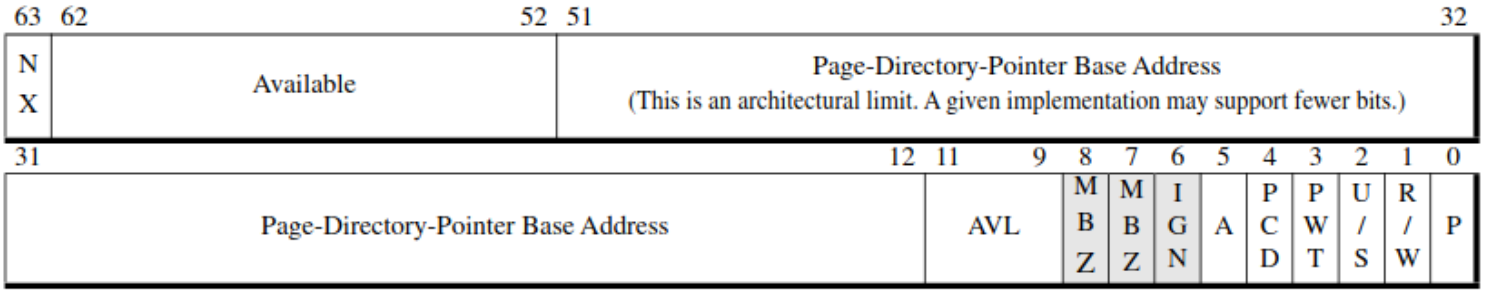


Figure 5-18. 4-Kbyte PML4E—Long Mode

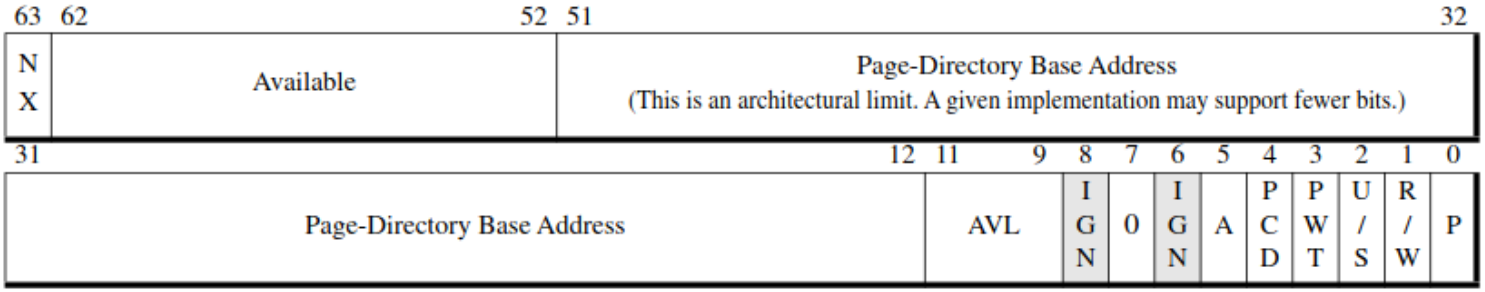


Figure 5-19. 4-Kbyte PDPE—Long Mode

* The picture is taken from
https://www.amd.com/system/files/TechDocs/24593.pdf

Example: x86-64

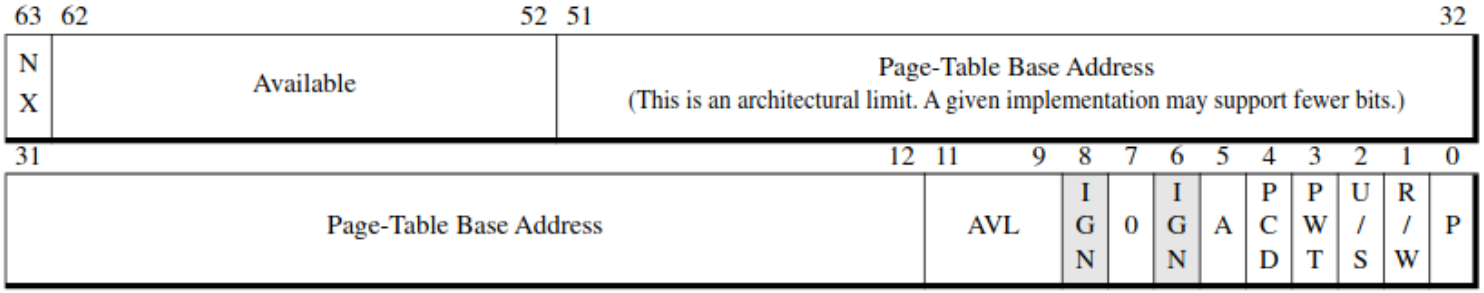


Figure 5-20. 4-Kbyte PDE—Long Mode

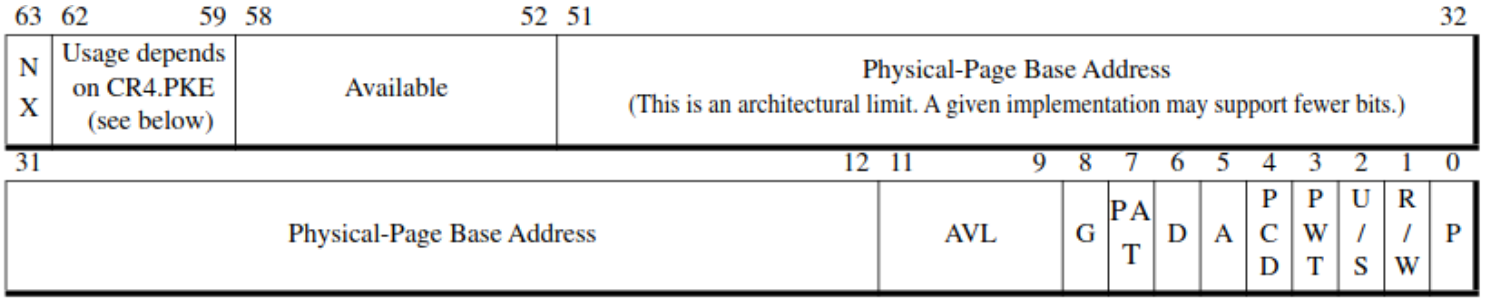


Figure 5-21. 4-Kbyte PTE—Long Mode

* The picture is taken from
https://www.amd.com/system/files/TechDocs/24593.pdf

Example: PTE

```
typedef unsigned long long u64; // a 64-bit integer type

struct page_pte {
    u64 present:1;        // Bit P
    u64 writable:1;       // Bit R/W
    u64 user_mode:1;      // Bit U/S
    ...
    u64 page_address:40; // 40+12 = 52-bit physical address (max)
    u64 avail:7;          // reserved, should be 0
    u64 pke:4;            // no MPK/PKE, should be 0
    u64 nonexecute:1;
};
```

Example: PTE

```
typedef unsigned long long u64; // a 64-bit integer type

struct page_pte {
    u64 present:1;        // Bit P
    u64 writable:1;       // Bit R/W
    u64 user_mode:1;      // Bit U/S
    ...
    u64 page_address:40; // 40+12 = 52-bit physical address (max)
    u64 avail:7;          // reserved, should be 0
    u64 pke:4;            // no MPK/PKE, should be 0
    u64 nonexecute:1;
};
// _page_ address, i.e. memory_address / 4096 (or 2^12)
p->page_address = 0xZZZZZZZZZZULL; // ULL is unsigned long long
p->writable = 1;
p->present = 1;
p->user_mode = 0;
...
p->avail = 0;
p->pke = 0;
p->nonexecute = 0;
```

Example: Initializing 4 GB

For 4GB, we reference 1048576 physical pages
Using 2048 pages for PTEs, 4 pages for PDEs, 1 for PDPE, 1 for PMLE4E
Total: 2054 pages = 8413184 bytes, align at the 4096 boundary!

PTE:

```
struct page_pte *p; … // Each entry is 8 bytes
for (size_t i = 0; i < 1048576; i++) { // 1048576/512 = 2048 pages
        // physical pages 0,1,2,3,…,1048575 (absolute address)
        p[i].page_addr = …
}
```

PDE:

```
struct page_pde *pd = (struct page_pde *) (p + 1048576);
for (size_t j = 0; j < 2048; j++) { // 2048/512 = 4 pages
        struct page_pte *start_pte = p + 512 * j;
        pd[j].page_addr = (u64) start_pte >> 12; // PTE page address
        …
}
```

PDPE:

Reference 4 PDEs (1 page), everything else is empty

PMLE4E:

Just one reference to PDPE; everything else is empty

Aligning Pages

Allocate more space: e.g., 8413184 + 4095

Align the allocated ‘base’:

```
(void *) (((size_t) base + 4095) & (~4095ULL))
```