OSTEP Chapter 20

ECE 3600, Fall 2022

Table of Contents

1. Linear Page Table
2. Segment Tables
3. Two-level Page Tables
4. 16KB Example
5. 16KB Example with Page Directory
6. Page Directory Example
7. Multi-level Page Tables
8. Opteron Page Tables
9. Exercises

1. Linear Page Table

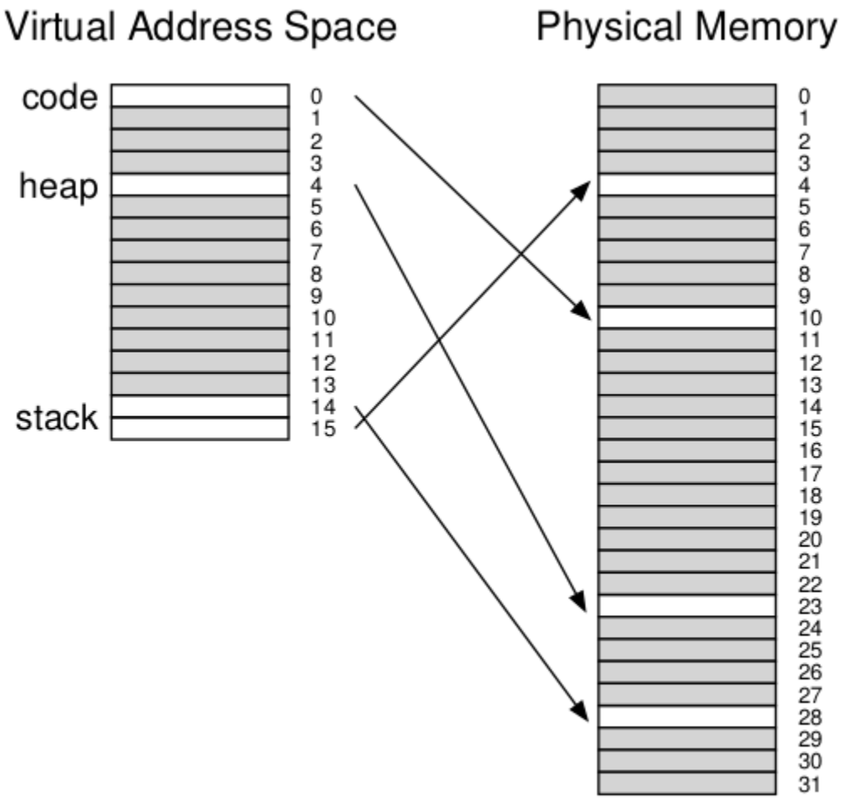


Figure 20.1: **A 16KB Address Space With 1KB Pages**

Most of the page table is unused, full of invalid entries.

PFN	valid	prot	present	dirty
10	1	r-x	1	0
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
23	1	rw-	1	1
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
-	0	—	-	-
28	1	rw-	1	1
4	1	rw-	1	1

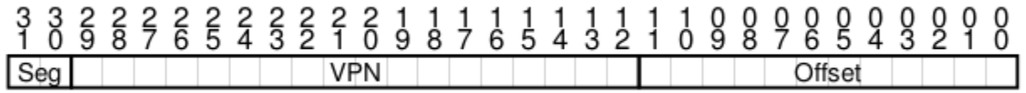
Figure 20.2: **A Page Table For 16KB Address Space**

2. Segment Tables

Base and bounds registers for each segment

Example: 32-bit virtual address space, 4KB pages, three segments: code, heap, stack

2-bit segment selector: 00 unused, 01 code, 10 heap, 11 stack



TLB miss handling:

```
SN = (VirtualAddress & SEG_MASK) >> SN_SHIFT

VPN = (VirtualAddress & VPN_MASK) >> VPN_SHIFT

AddressOfPTE = Base[SN] + (VPN * sizeof(PTE))
```

specify SEG_MASK, SN_SHIFT, VPN_MASK, VPN_SHIFT: _____

3. Two-level Page Tables

page directory: **PDE** = page directory entry, **PFN** = page frame number

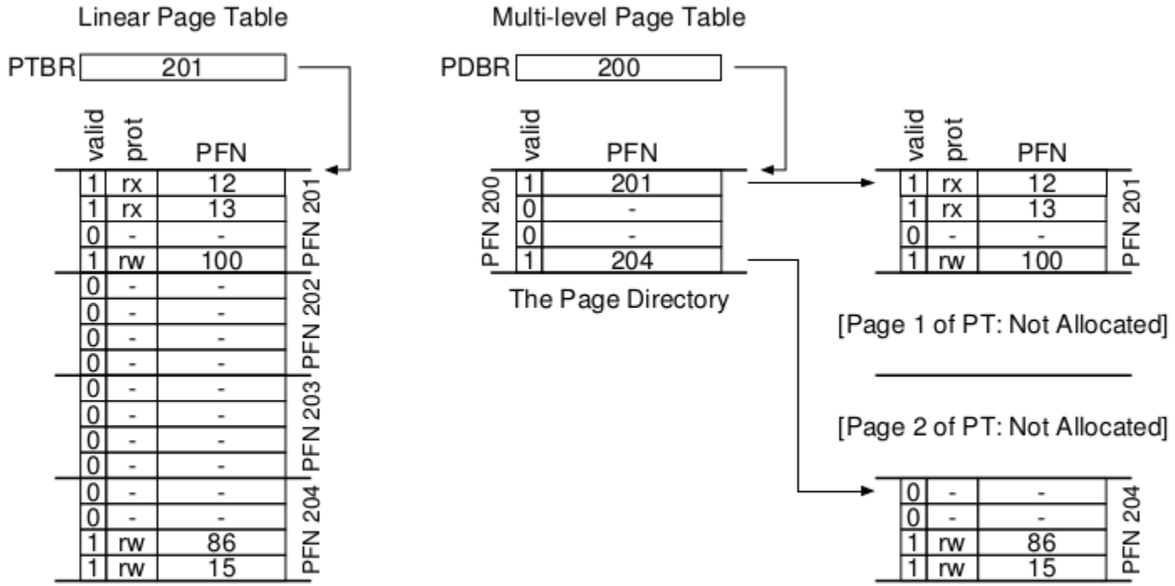


Figure 20.3: **Linear (Left) And Multi-Level (Right) Page Tables**

4. 16KB Example

Address space size 16KB with 64-byte pages: 14-bit virtual address = 8 bits VPN + 6 bits offset

Linear page table would have 2^8=256 entries

Example with virtual pages 0 and 1 used for code, 4 and 5 for the heap, 254 and 255 for the stack:

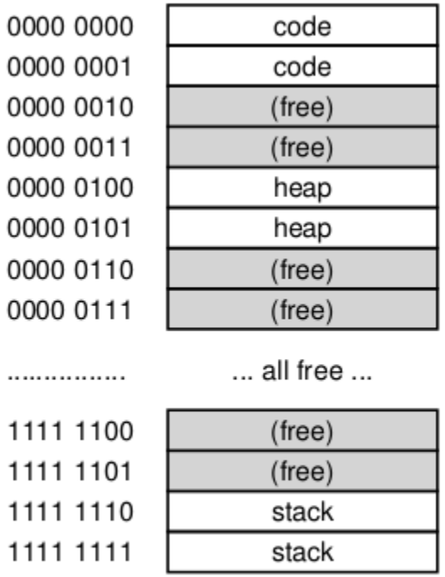


Figure 20.4: **A 16KB Address Space With 64-byte Pages**

5. 16KB Example with Page Directory

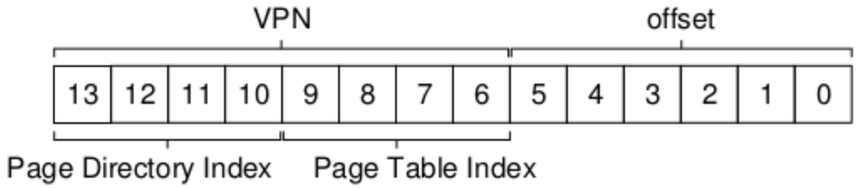
256 page table entries * 4 bytes each = 1 KB = 16 * 64-byte pages with 16 PTEs per page

4 bits to index one of the 16 entries in the page directory:

```
PDEAddr = PageDirBase + (PDIndex * sizeof(PDE))
```

4 bits to index one of the 16 PTEs:

```
PTEAddr = (PDE.PFN << SHIFT) + (PTIndex * sizeof(PTE))
```



6. Page Directory Example

PDIndex 0, 15 valid; PFN 100 PTIndex 0, 1, 4, 5 valid; PFN 101 PTIndex 14, 15 valid

Page Directory		Page of PT (@PFN:100)			Page of PT (@PFN:101)		
PFN	valid?	PFN	valid	prot	PFN	valid	prot
100	1	10	1	r-x	—	0	—
—	0	23	1	r-x	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	80	1	rw-	—	0	—
—	0	59	1	rw-	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	55	1	rw-
101	1	—	0	—	45	1	rw-

Figure 20.5: **A Page Directory, And Pieces Of Page Table**

VA = 0x3f80 = 1111 1110 000000 = PDIndex + PTIndex + offset

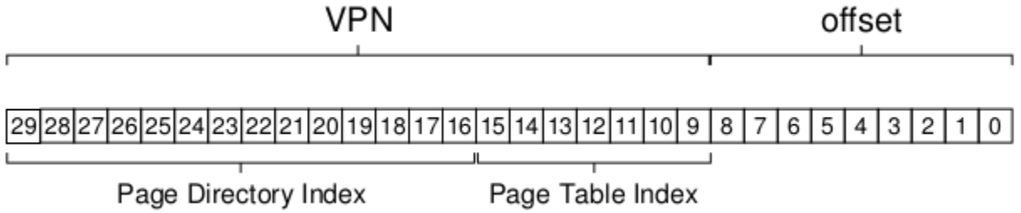
PDIndex = 15 --> PFN 101; PTIndex = 14 --> 55 (00110111); PhysAddr = 00110111 000000 = 3520

7. Multi-level Page Tables

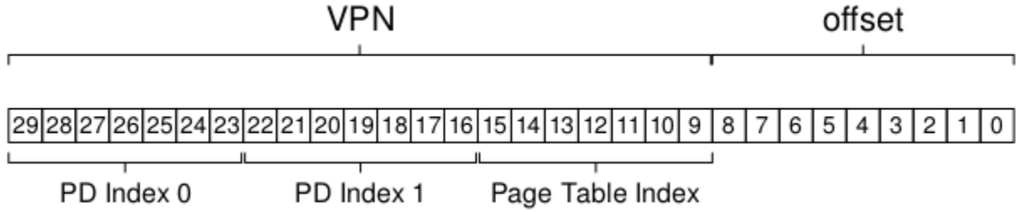
30-bit virtual address space, 512 byte page = 21-bit virtual page number + 9-bit offset

PTE size 4 bytes --> 128 PTEs per page --> 7-bit PTIndex --> 14-bit PDIndex

2^{14} 4-byte page directory entries --> 128 pages



Split page directory into two levels:



2^{7} 4-byte page directory entries --> 1 page

8. Opteron Page Tables

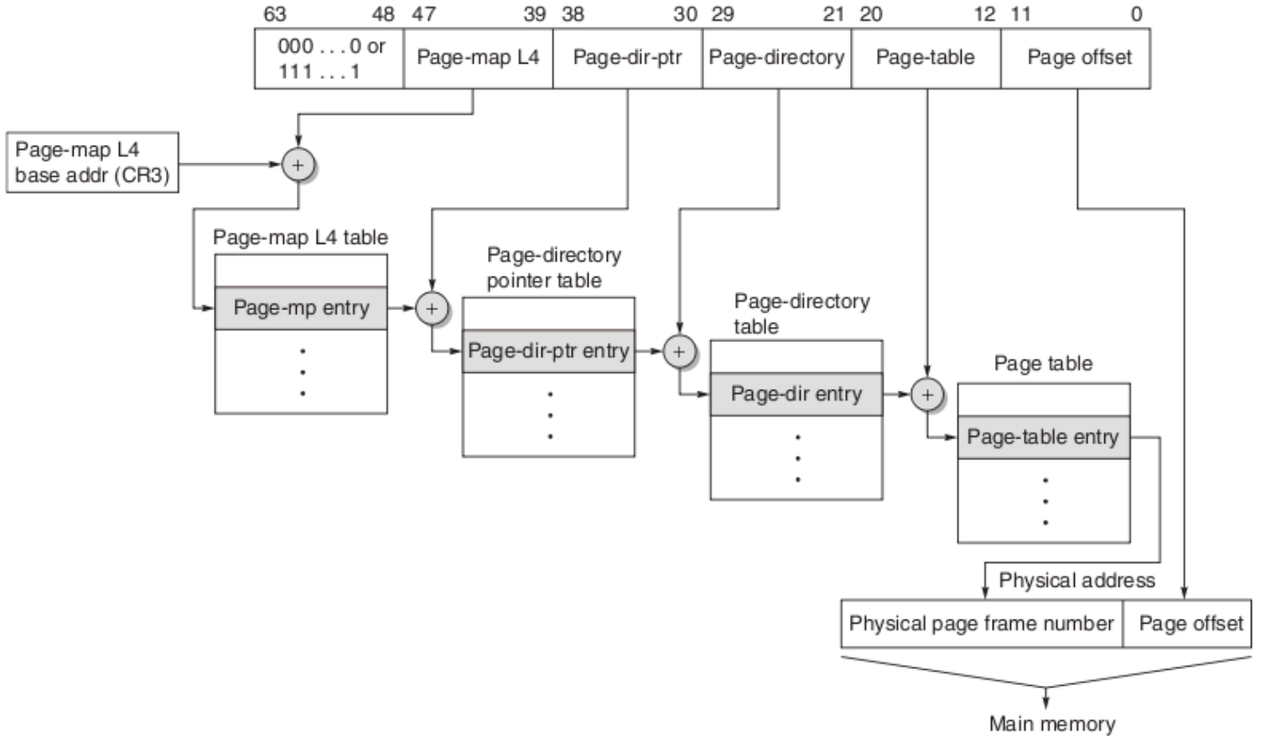


Figure B.27 The mapping of an Opteron virtual address. The Opteron virtual memory implementation with four page table levels supports an effective physical address size of 40 bits. Each page table has 512 entries, so each level field is 9 bits wide. The AMD64 architecture document allows the virtual address size to grow from the current 48 bits to 64 bits, and the physical address size to grow from the current 40 bits to 52 bits.

The Opteron uses 48-bit virtual addresses and 40-bit physical addresses.

The upper 16 bits of the virtual address are just the sign extension of the lower 48 bits.

(from Hennessy & Patterson)

9. Exercises

Exercises from the book using paging-multilevel-translate.py:

1. With a linear page table, you need a single register to locate the page table, assuming that hardware does the lookup upon a TLB miss. How many registers do you need to locate a two-level page table? A three-level table?

2. Use the simulator to perform translations given random seeds 0, 1, and 2, and check your answers using the -c flag. How many memory references are needed to perform each lookup?

3. Given your understanding of how cache memory works, how do you think memory references to the page table will behave in the cache? Will they lead to lots of cache hits (and thus fast accesses?) Or lots of misses (and thus slow accesses)?